



Rumoca

Modelica as a Universal Algebraic Frontend via a Rust-Native Compiler

Micah K. Condie, Abigaile Woodbury, Thomas Meschede, Joel A. E. Andersson, James M. Goppert

Purdue University · Xyntopia LLC · FMIOPT AS

American Modelica Conference 2026

Outline



Purpose and History

Why one Modelica source should feed many tools, why it suits AI, and how Rumoca got here



Design

Features, compiler architecture, templates, MSL coverage



Use Cases

Browser, multi-backend export, eFMI flight, FastDyn SITL, SLAM

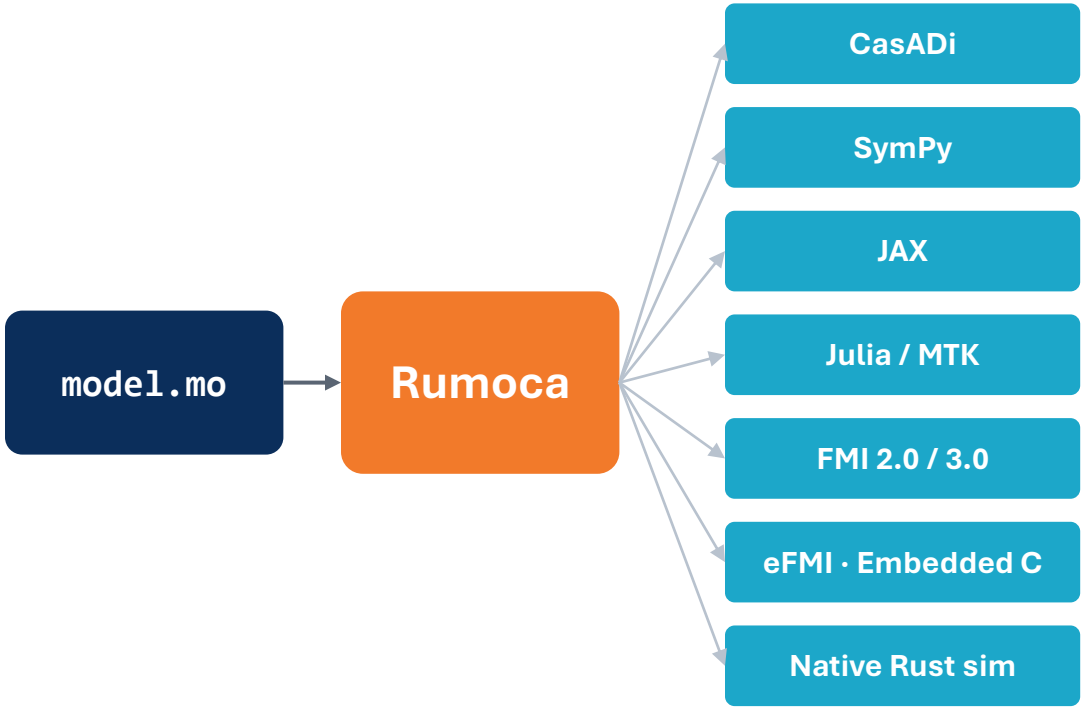


Summary and Outlook

What Rumoca delivers today and where it goes next

Purpose of Rumoca

- **Modelica as a universal algebraic frontend**
  - **RU**st **MO**delica to **C**omputer **A**lgebra
- Optimization, differentiable simulation, SciML, and system analysis live in a variety of non-Modelica tools: CasADi, JAX, Julia, SymPy
- Today the same model is written multiple times: Modelica to simulate, CasADi to optimize, SymPy to analyze, with no guarantee they agree
- Rumoca: write it once in declarative Modelica, emit it in each tool's native idiom



A Brief History of Rumoca

1

Pymoca

Predecessor

Python + ANTLR Modelica translator to CasADi. Accessible, but slow parsing and untyped ASTs limit large models.

2

Rumoca as a Translator

v0.1–0.7 · 2024–25

Rust rewrite with typed ASTs and Jinja templates. CasADi + SymPy backends. Modelica Conference 2025: 3 example models, external compiler needed to flatten complex ones.

3

AI-Assisted Full Compiler

v0.8 · Mar 2026

Resolution, type checking, instantiation, flattening, DAE, structural analysis. Humans set phase contracts; agents implement inside them. MSL parity gate in CI.

4

Jacobians, SIL

v0.9–0.10 · 2026

Exact automatic-differentiation Jacobians, realtime SIL with Cerebri, WASM playground, eFMI/GALEC export.

Why Modelica Suits AI

1 Plain text is AI-native

A model is a file: read, diffed, version-controlled, generated by a script. An LLM reads the equations and proposes edits in the same medium. A block diagram has to be serialized first.

```
model BouncingBall
  parameter Real e = 0.8;
  parameter Real g = 9.81;
  Real h(start = 1.0), v;
equation
  der(h) = v;
  der(v) = -g;
  when h <= 0 and v < 0 then
    reinit(v, -e*pre(v));
  end when;
end BouncingBall;
```

2 25 years of models to ground on

Modelica's open corpus, led by the Modelica Standard Library, gives LLMs real, reviewed models to learn from and be checked against.

25+
years of Modelica

1,288
MSL component models

BSD-3
open license

Electrical | Mechanical | Thermal | Fluid | Control

Domains covered by the MSL

Rumoca Features

Modelica as universal frontend

CasADi (MX / SX)	SymPy	JAX
Julia / MTK	FMI 2.0 / 3.0	eFMI (GALEC)
Embedded C	JavaScript	ONNX
SymForce	Base Modelica	Custom Jinja

Every backend is a Jinja template over the compiler IR. New target = new template.

Ships as CLI, Python wheel, npm package, VS Code extension · Apache-2.0

Simulation

Native Rust runtime: RK45 and Diffsol (BDF / ESDIRK) solvers, exact AD Jacobians, Cranelift JIT, realtime SIL over UDP / WebSocket

VS Code extension

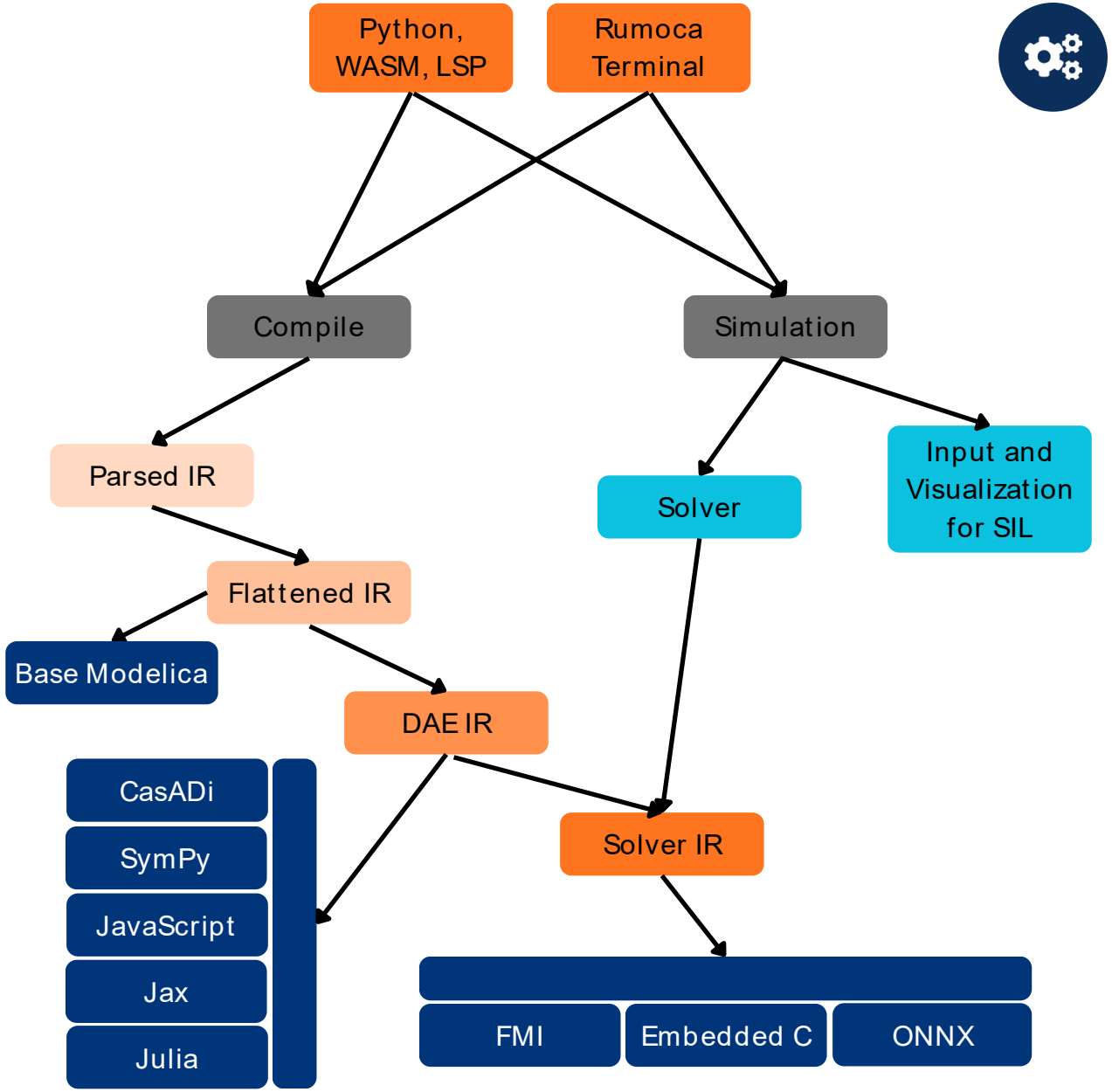
Language server with live diagnostics, autocomplete, go-to-definition; one-click simulation to an HTML report; Three.js 3D view

WASM deployment

The whole compiler runs in the browser: zero-install playground and standalone HTML simulators

Rumoca Architecture

- **Rust**: every phase is its own crate; phases talk only through typed IRs
- **Parol** LL(k) parser → Parsed IR
- **Flatten** → Flattened IR, connects resolved
- **DAE IR**: hybrid DAE per MLS App. B
- **Structural prep**: matching, BLT, tearing, dummy derivatives
- **Solver IR**: register opcodes + forward-mode AD, run interpreted or Cranelift-JIT
- **Templates** can read any IR to generate backends



Backends Are Jinja Templates

1 Modelica model

```
model SympyDecay
  Real x(start=1);
  parameter Real k = 0.5;
equation
  der(x) = -k*x;
end SympyDecay;
```

2 Template (casadi-ode, excerpt)

```
{%- elif op.kind == "LoadY" %}
  __r{{ op.dst }} = x[{{ op.index }}]
{%- elif op.kind == "LoadP" %}
  __r{{ op.dst }} = P[{{ op.index }}]
{%- elif op.kind == "Unary" %}
  __r{{ op.dst }} = {{ unary(op.operator, op.arg) }}
{%- elif op.kind == "Binary" %}
  __r{{ op.dst }} = {{ binary(op.operator, op.lhs, op.rhs) }}
```

3 Generated CasADi

```
__r0 = P[0]
__r1 = x[0]
__r2 = (__r0 * __r1)
__r3 = (-__r2)
out[0] = __r3
xdot = ca.vertcat(*out)
rhs = ca.Function('rhs', ...)
```

*Rumoca lowers der(x) = -k*x to Solve IR ops; the template only prints them.*

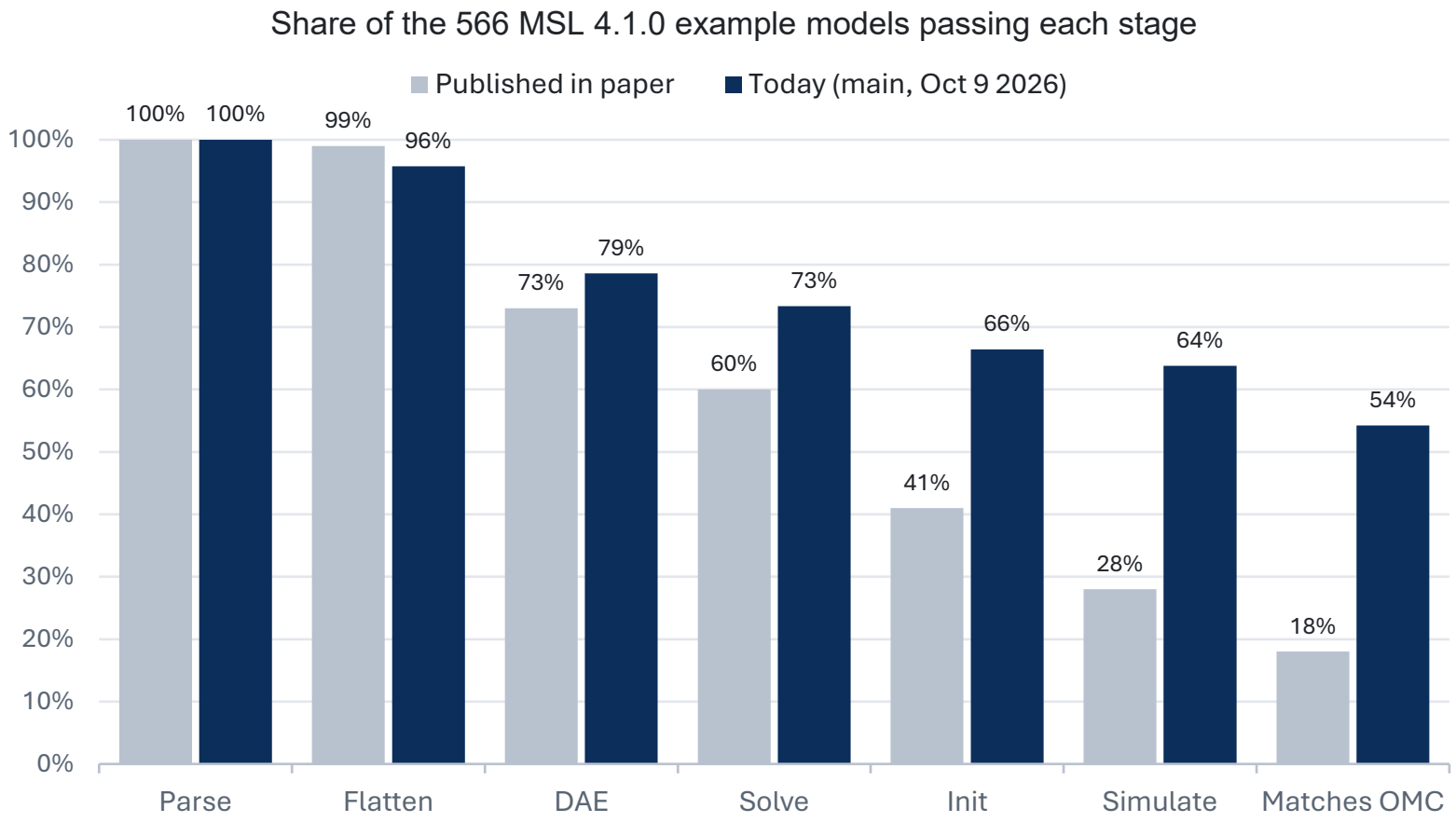
Write your own: a 3-line template

```
{% for v in dae.variables -%}
{{ v.name }} : {{ v.role }}
{% endfor -%}
```

```
x : state
k : parameter
```

- **A target** is a target.toml naming the IR it reads (flat, DAE, solve, FMI, eFMI) plus its .jinja files
- **Compiler owns semantics,** templates only print; an unsupported op calls fail() instead of guessing
- **All 17 built-in targets** (CasADi, JAX, C, Rust, FMI, eFMI…) are written this way

MSL Coverage: Paper vs. Today



28% → 64%

simulate end to end
of all 566 example models

102 → 307

match OpenModelica traces
3× more, under a stricter test

0

deviating traces
among the 307 compared

Paper: MSL 4.1.0 vs OMC 1.26.7. Today: CI quality gate on main after PR #417, same 566 models.

Use Cases

1. **Simulation in browser**
   WASM editor and simulation, nothing to install

   Zero install

2. **CasADi and SymPy export in VS Code**
   Editor plus universal frontend: one model, four backends

   Universal frontend

3. **eFMI on a fixed wing**
   Modelica control model flown as embedded code

   Embedded systems

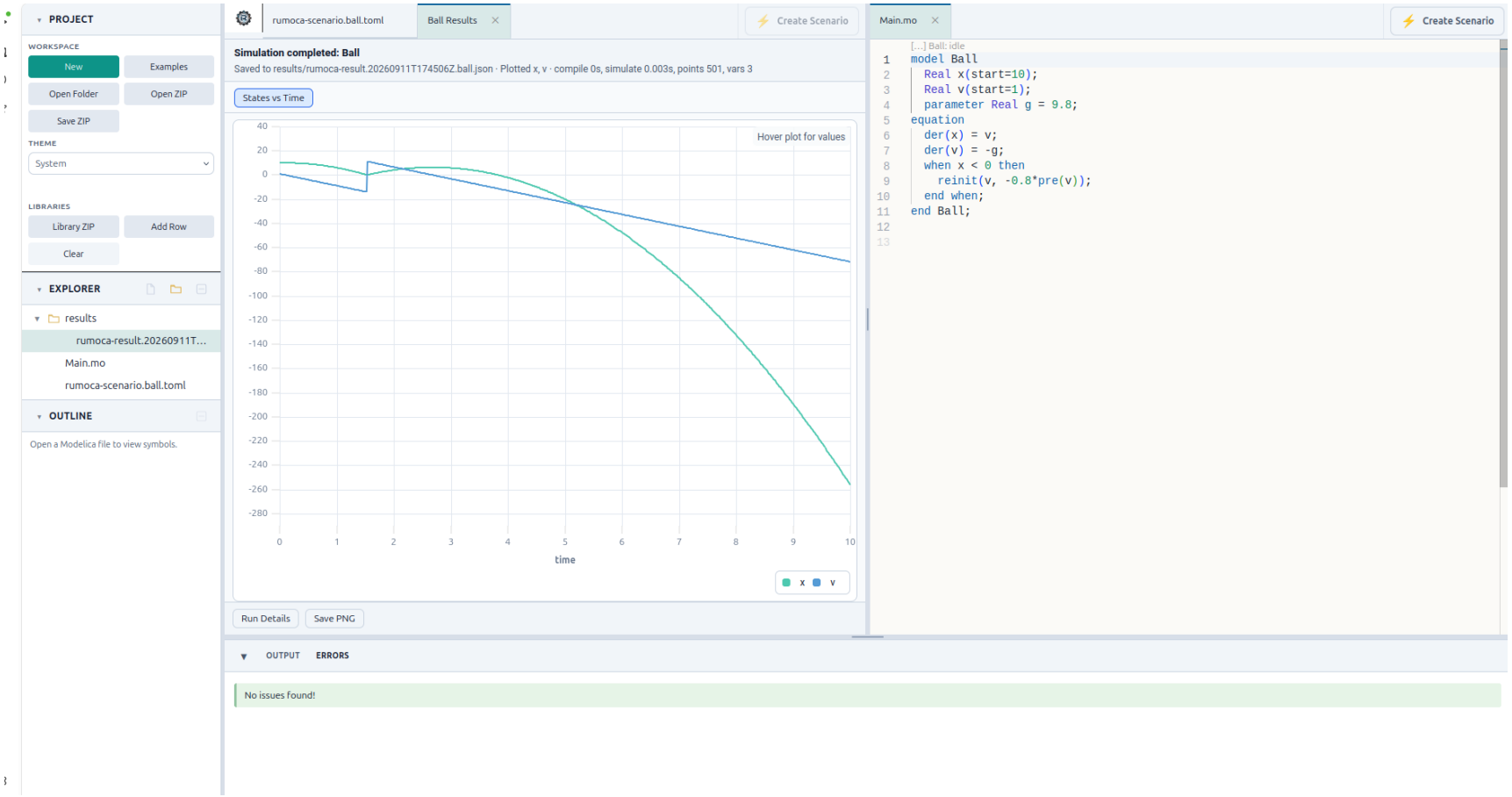
4. **FastDyn quadrotor**
   Rumoca plant model inside a fast ArduPilot SITL toolchain

   Other toolchains

5. **SLAM in the browser**
   A full RGB-D SLAM pipeline written in Modelica

   Rumoca extreme

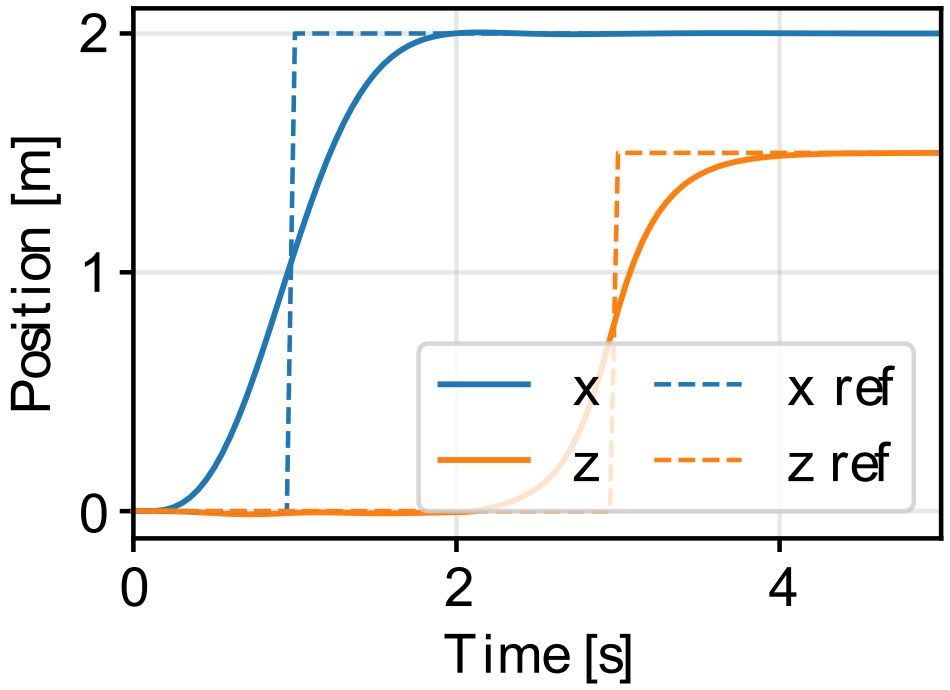
Simulation in Browser



Try it now
cognipilot.github.io/rumoca

Whole compiler runs client-side via WebAssembly · no server, no login · models stay on your machine

CasADi / SymPy Export Example

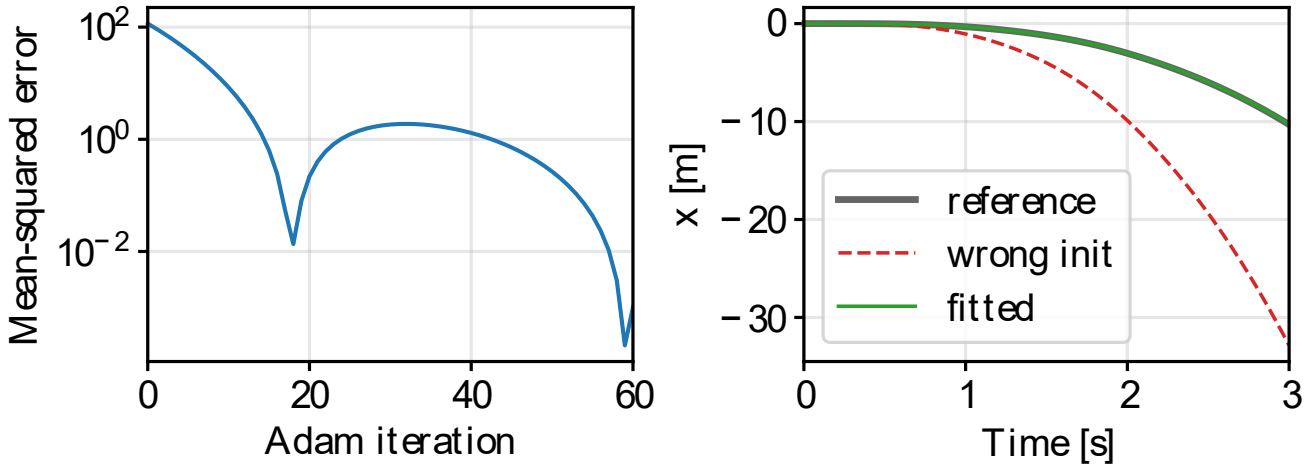


Linear MPC on the Rumoca-emitted CasADi model

One 20-line PVTOL model, four artifacts

CasADi MX | SymPy | JAX | FMI 3.0

- **CasADi:** linear MPC at 20 Hz, mean QP solve < 10 ms
- **SymPy:** closed-form A, B; controllability rank 6
- **Cross-check:** CasADi vs SymPy Jacobians agree to 10^{-14}
- **JAX:** recovers m, J within 0.1% / 0.4% **FMI:** ~200× realtime



JAX parameter identification through a differentiable rollout

eFMI on a Fixed Wing



FixedWing.mo
Rate-command / attitude-hold controller in Modelica

↓

Rumoca --target galec
eFMI Algorithm Code (GALEC)

↓

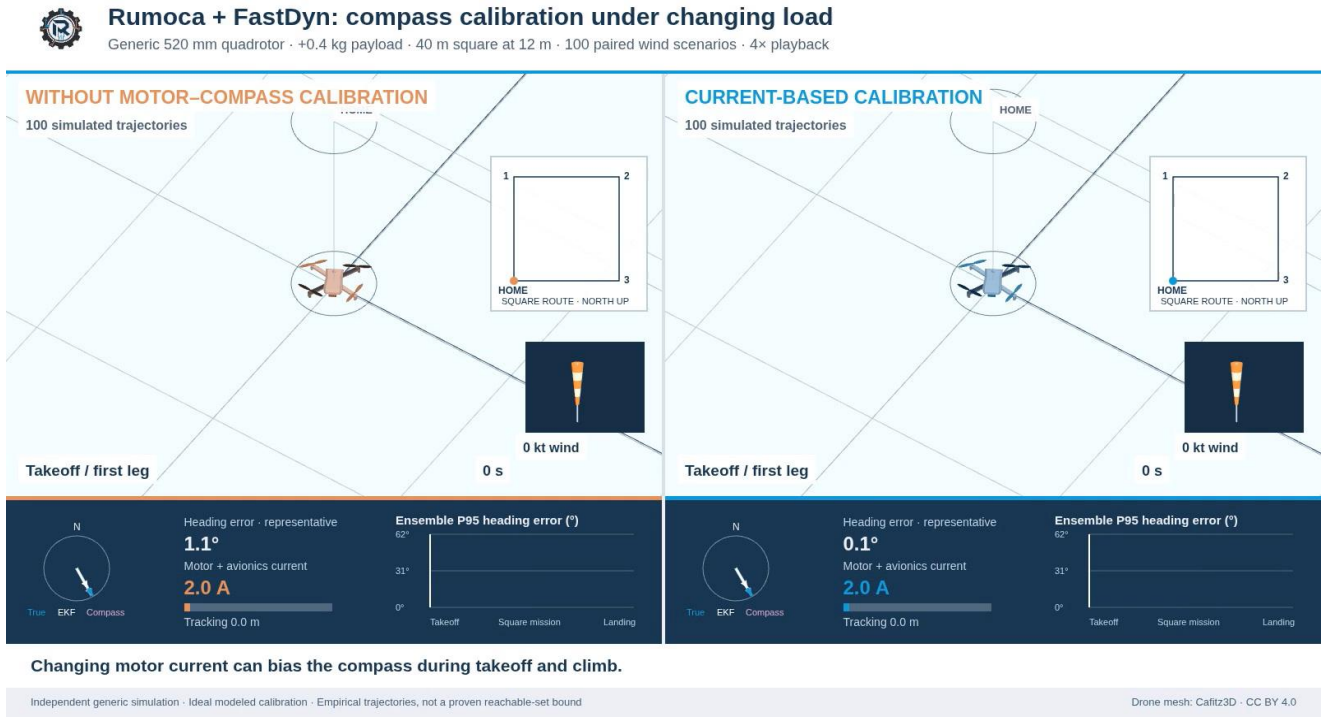
eFMI Production Code
C for the embedded target

↓

Flight controller
Runs on the aircraft

The same Modelica source runs as a 6-DOF SIL plant with a Three.js viewer before it ever flies.

FastDyn Quadrotor



ArduPilot firmware
Unmodified flight code

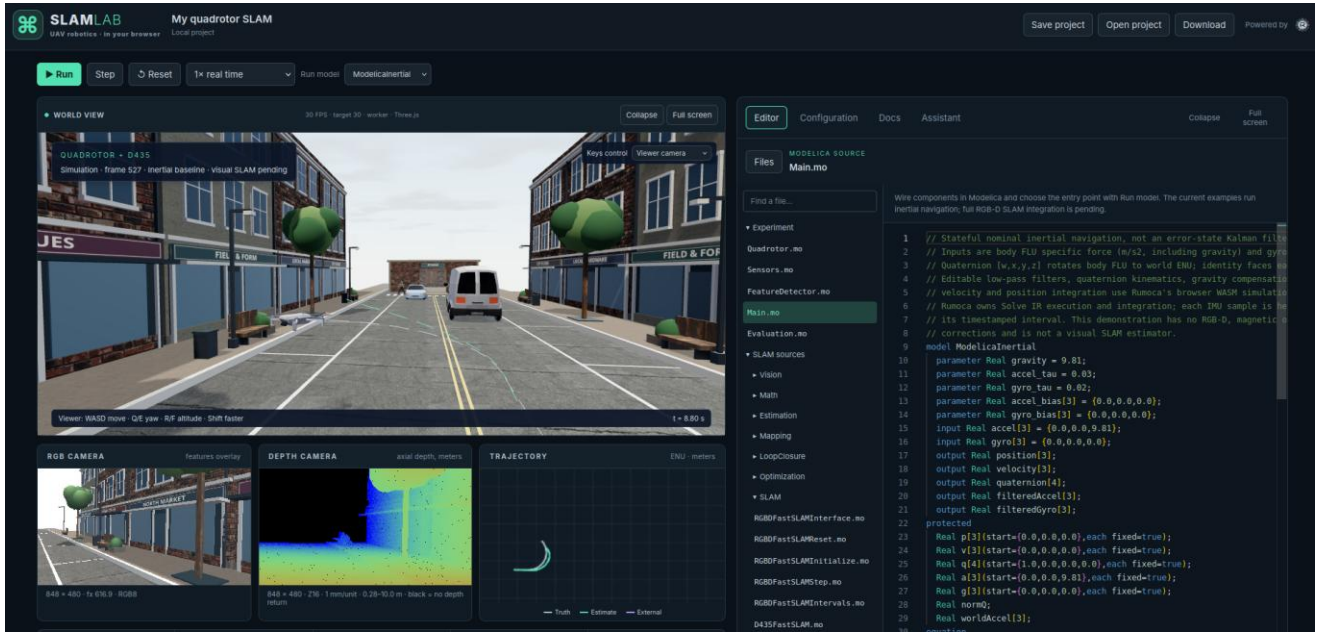
FastDyn
Runs the firmware much faster than realtime

Rumoca
Quadrotor plant dynamics compiled from Modelica

Fast SITL simulation (10x)

Rumoca integration into other tools

SLAM in the Browser



- **RGB-D SLAM written in Modelica**
- Compiled by Rumoca to WebAssembly and run entirely client-side
- **Pushes the compiler far past MSL:** image-sized arrays, stencil kernels, function-heavy code
- Same compiler and Session API as the playground

Summary

1

One source, many backends

CasADi, SymPy, JAX, Julia, FMI, eFMI, C from a single .mo, and backends agree to machine precision

2

A real Modelica compiler

Explicit IR phases, structural analysis, exact AD Jacobians, MSL parity tracked in CI

3

Runs everywhere

CLI, Python, npm, VS Code, the browser via WASM, realtime SIL

4

Fits real toolchains

eFMI flight code, ArduPilot + FastDyn SITL, SLAM in a browser tab

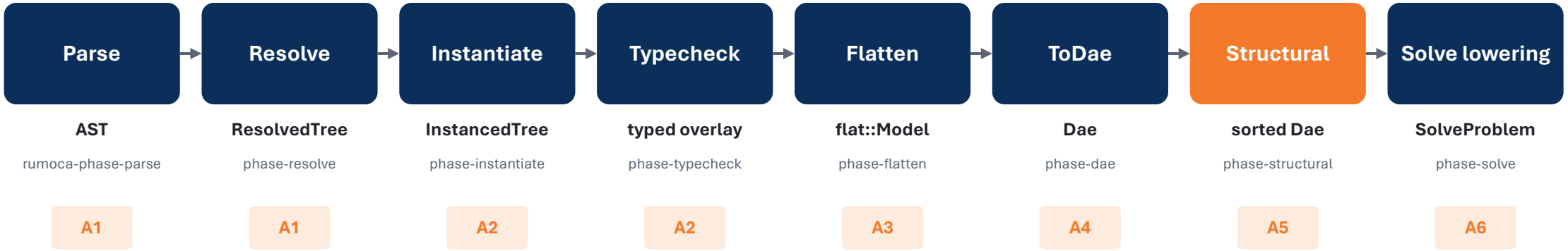
github.com/CogniPilot/rumoca · Apache-2.0 · pip install rumoca

Outlook

Coverage	Raise end-to-end MSL simulation. Built-in tracing with span preservation, so compile and simulation errors map back to source lines.
Backends	Embedded C to production. Broader FMI 3.0. JAX, Julia, and ONNX up to CasADi / SymPy maturity.
Verification	Formally verify compiler semantics: correct IR construction and semantic lowering.
Community models	MSL measures language coverage, not real use. Contribute optimization and ML models that already simulate elsewhere as reference cases.

Try it: cognipilot.github.io/rumoca · feedback and models welcome on GitHub issues

Appendix: Compiler Internals



- **One crate per phase.** 56 workspace crates in 6 tiers; IR crates are pure data and phases may not depend on each other.
- **One Session API.** CLI, VS Code (LSP), WASM and Python all drive the same Session; per-stage caches keyed by dependency fingerprints.
- **Reject, never mistranslate.** Each phase owns an error-code family (EP, ER, ET, EI, EF, ED, ES, EL, EC) and refuses unsupported features with a diagnostic.

Execute: interpreter · Cranelift · WASM

Codegen: MiniJinja targets

A6 Solve IR · A7 Codegen · A8 Numerics · A9 Verification · A10 Speed vs OMC

A1 · Parsing and Name Resolution

Parsing (Parol)

- **LL(k) with k ≤ 3:** only 2 of 312 decisions need k = 3; grammar is modelica.par, Modelica 3.7-dev, 519 productions
- **Never fails:** returns a tree plus errors; re-parses with a virtual semicolon (up to 32 passes) to report many errors at once
- **Editor-friendly recovery:** a hand-written fallback parser keeps class headers and the outline alive while you type
- **Spans everywhere:** byte ranges tagged with a stable SourceId, carried AST → Flat → DAE → Solve
- **Same AST powers** the formatter and linter; the parser is fuzzed

Name resolution (MLS §5)

- **Scope tree:** Global, Package, Class, Encapsulated, Function, ForLoop
- **Lookup order:** members → imports → inherited members → parent; encapsulated blocks upward lookup; conflicting bases stop lookup
- **Identity is a DefId** after resolve, never a name string
- **Semantic checks live here too:** ~134 ER codes covering clocks, streams, state machines, class restrictions, builtins
- **Libraries:** --source-root plus MODELICAPATH; package.mo layout validated, within clauses merged

A2 · Instantiation and Type Checking

Instantiation (MLS §7)

- **extends** with a per-base cache that handles diamond inheritance; selective extends with break
- **Modifications:** outer overrides inner, each evaluated in the scope where it was written; final and each enforced
- **redeclare / replaceable** checked against constraining types
- **Conditional components** need translation-time parameters; disabled ones add nothing
- **Component arrays:** a homogeneous array is instantiated once and replicated, provably identical to element-wise expansion
- **inner / outer** lookup per §5.4

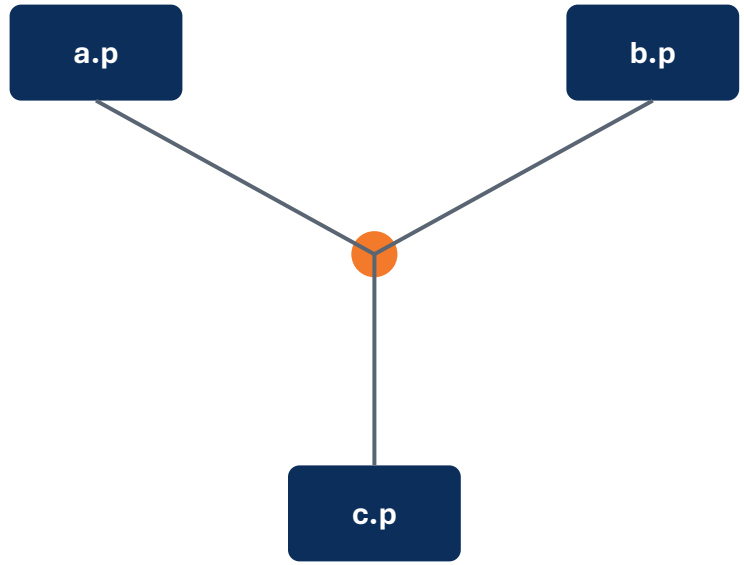
Type checking

- **Runs after instantiation:** array sizes can depend on modifiers (§10.1)
- **Dimensions:** evaluates dimension expressions, infers ':' from bindings, marks structural parameters
- **Expressions and equations:** type compatibility, builtin-call signatures
- **Variability** ordering (constant ≤ parameter ≤ discrete ≤ continuous) and causality rules
- **Units:** unit strings are validated against the §19 grammar only; no dimensional analysis yet

A3 · Flattening and Connections

What flatten produces

- **connect equations:** potential variables → n−1 equalities; flow variables → one sum-to-zero (+ inside, − outside connectors)
- **Stream connectors:** inStream / actualStream eliminated while connection sets still exist
- **Overconstrained connectors:** Connections.root / branch / potentialRoot build a spanning forest; broken edges become equalityConstraint(...)
- **Expandable connectors:** partial; identically declared buses work, member-union augmentation is rejected explicitly
- **Arrays stay compact:** for-equations and array equations become structured equation families over index domains instead of being unrolled
- **Function-valued arguments** removed by specializing the callee




```
a.p.v = b.p.v = c.p.v
a.p.i + b.p.i + c.p.i = 0
```

A4 · Hybrid DAE (MLS Appendix B)

$$v = [\ p\ ;\ t\ ;\ \dot{x}\ ;\ x\ ;\ y\ ;\ z\ ;\ m\ ;\ pre(z)\ ;\ pre(m)\ ]$$

`fx`	`fz`	`m := fm`	`fc`
continuous residual	discrete Real update	discrete-valued assignment	event conditions
B.1a	B.1b	B.1c	B.1d

Variables and checks

- **Roles:** Parameter, Constant, Input, State, Algebraic, Output, DiscreteReal, DiscreteValue
- **Balance** checked here (ED001) for non-partial models
- **Valid by construction:** Dae::construct requires provenance on every node; schema-versioned

Temporal operators are gone after ToDae

- **pre(z)** is a typed coordinate; edge and change become equations over current and pre values
- **reinit** becomes a guarded state update; assert / terminate become event actions
- **Clocks:** periodic, interval-counter and event clocks; sub/super/shift/back-sample

A5 · Structural Analysis



Sorting

- **Alias:** union-find over a = ±b with sign parity; representative chosen by StateSelect, fixed, start, nominal
- **Incidence:** exact scalar incidence from checked expressions; provably-zero terms dropped; array families stay structured
- **Matching:** Kuhn augmenting paths on the main path; Hopcroft–Karp for sortability screening
- **BLT:** Tarjan SCC with an explicit stack; self-contained array families form one block

Solving the hard blocks

- **Tearing:** greedy Cellier-style; two plans (causal unlocks vs. degree), fewer tears wins
- **Proven causality:** a causal step without a nonzero-coefficient proof is promoted to a tear
- **Index reduction:** Pantelides-style dummy derivatives; index-1 lift complete, higher index an explicit subset
- **Constraints kept, not lost:** holonomic constraints remain as manifold residuals with runtime projection
- **State selection:** StateSelect.always / prefer; up to 4 alternate charts switched at runtime

A6 · Solve IR and Automatic Differentiation

SolveProblem

- **Partitions:** continuous, initialization, discrete, events, clocks; versioned and re-validated on load
- **Continuous system:** implicit RHS, residual, derivative RHS, manifold residual, plus algebraic projection plans
- **Instructions:** register programs (~50 ops: arithmetic, select, matmul, linear solve, pure calls, tables) migrating to typed SSA with per-op provenance
- **Optimizations stay IEEE-exact:** literal folding, x·1, x/1, −(−x); dead-constant pruning; bounded value reuse. No global CSE yet

Derivatives

- **Forward mode:** JVPs generated from the Solve IR, keeping tensor nodes intact
- **Reverse mode:** VJPs including the linear-solve adjoint
- **Sparsity:** index-set propagation plus distance-2 coloring (Curtis–Powell–Reid); mass matrix Identity / Diagonal / Sparse
- **New:** forward trajectory sensitivities, adjoint objective gradients, A/B/C/D linearization (rumoca --inspect)

A7 · Execution and Code Generation

Execution engines

Interpreter
Reference engine; differential oracle for the others

Cranelift JIT
Native default when the model is admitted

WASM
Browser and playground builds

MLIR / CUDA / WGSL
Experimental kernels

Template targets (MiniJinja)

Validated

- c-ode
- rust-ode
- rust-fixed-ode
- galec
- galec-production
- embedded-c-galec

Checked

- casadi-ode
- jax-ode
- fmi2
- fmi3
- cuda-ode
- mlir
- dae-modelica
- flat-modelica

Experimental

- wgsl-ode
- fmi-ls-wasm
- base-modelica

- **Each target** is a target.toml plus templates that declare which IR they read (flat, DAE, solve, FMI, algorithm code)
- **FMI:** one source FMU, Model Exchange + Co-Simulation; CS steps with error-controlled DOPRI5 substeps
- **eFMI:** DAE → GALEC Algorithm Code (.alg) → Production Code C

A8 · Numerics and Runtime

Integration and initialization

- **RK45:** Dormand–Prince 5(4), FSAL, dense output
- **Diffsol:** variable-order BDF with sparse LU (faer)
- **auto:** BDF unless directional derivatives are unavailable, then RK; chosen once, never switched mid-run
- **Initialization (MLS §8.6):** der(x) replaced by its matched RHS; fixed=true pinned, fixed=false are guesses
- **Newton:** scaled, damped, backtracking line search, per-unknown trust region; homotopy λ-continuation 0 → 1
- **Torn blocks:** reduced Newton over tear variables with exact back-substitution; dense/sparse fallback, counted and reported

Events and SIL

- **Zero crossings** located on the integrator's dense output with Illinois refinement
- **Event iteration** up to 256 passes; reinit either restarts or preserves integrator history
- **Clocks:** exact rational clock lattice for periodic schedules
- **Pacing:** as fast as possible, realtime, or lockstep with an external autopilot
- **I/O:** FlatBuffers signal frames over UDP, WebSocket or Zenoh; gamepad and keyboard input

A9 · Verification and Quality Gates

470
MLS contracts tracked
421 implemented

586
contract test cases
327 reject · 259 accept

~10.5k
unit tests
across the crates

566
MSL examples
in the parity gate vs OMC

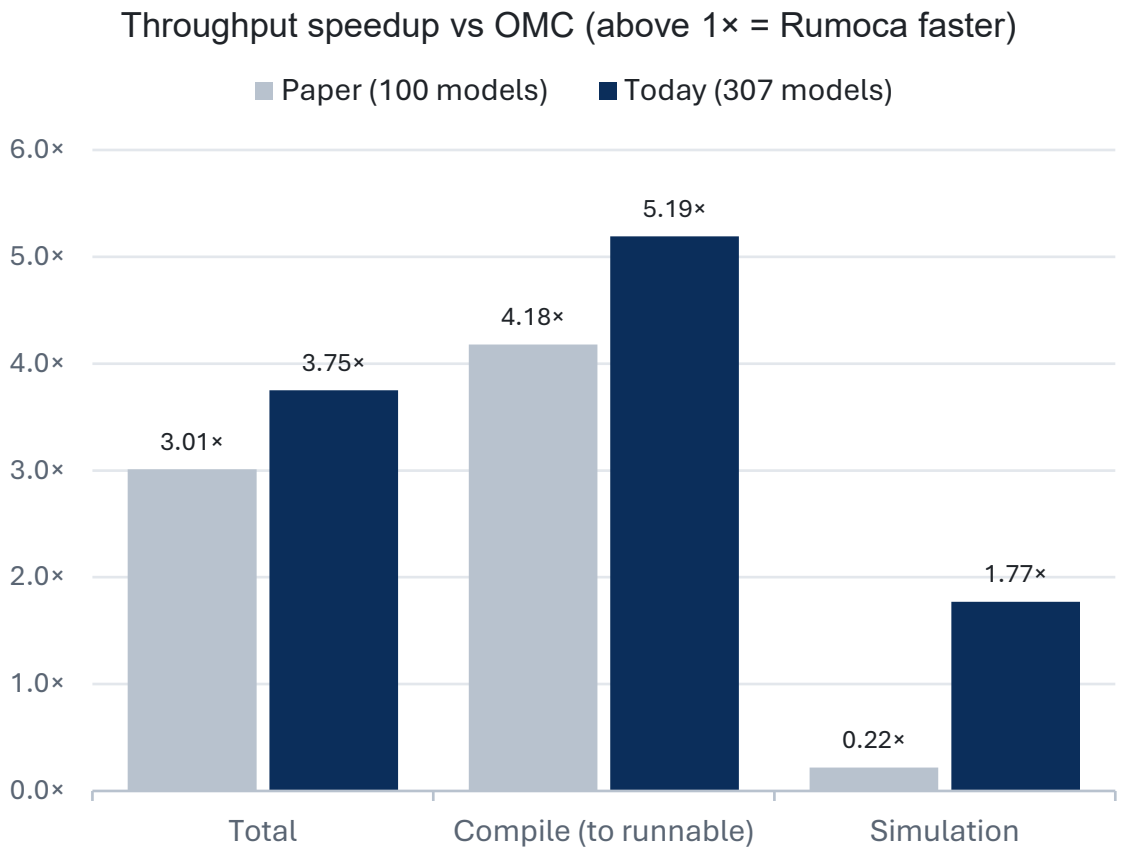
Gates

- **MSL parity:** stage counts ratcheted against a promoted baseline; traces compared to OMC per channel
- **Reference interpreter** for differential testing of every backend
- **Architecture tests** enforce crate boundaries and the Session-only API

Toward formal verification

- **Specs as contracts:** numbered SPEC documents define each phase; AI agents work inside them
- **Kani:** first proof harness (event-iteration history)
- **Lean:** proposed roadmap for verified semantics (SPEC_0052)

A10 · Speed vs. OpenModelica



	Paper	Today
Models timed	100	307
Median speedup, total	4.79×	15.33×
Median speedup, compile	n/a	18.45×
Median speedup, simulation	n/a	4.24×
Where Rumoca is slower	Total: 250+ eqn Sim: 50+ eqn	Total: 7 models Sim: 44 models

Caveat: both timings use only models where the two tools agree, which favors Rumoca. Each model timed once on a single host. FMU-path timing is not measured in CI.