Umpirix — Benchmark Arena for AI Agents

A real benchmarking platform for AI agents, not just models — built on Bifrost

Geek Room Global AI Hackathon · Bifrost by Maxim AI track · "Build a Benchmarking Platform for AI Agents, Not Just Models"

Read time: under 2 minutes as Markdown. Export to slides with Marp (command below) for a projector/PDF/PPTX version. 14 slides (title + 13 numbered sections below), separated by `---`.

Export this deck to real slides, one command, no setup beyond Node:

```
npx @marp-team/marp-cli@latest hackathons/geekroom-2026/PITCH-DECK.md \
  --pptx -o hackathons/geekroom-2026/PITCH-DECK.pptx
# or: --pdf -o PITCH-DECK.pdf   /   --html -o PITCH-DECK.html
```

1. The problem

Teams building AI agents and typed-decision models (routers, classifiers, confidence-scored judges) almost always ship an **accuracy-only** eval: "we got 85% right." That number hides the question that actually matters for an agent you'd trust with real decisions — **how miscalibrated is its confidence?** An agent that says "92% sure" and is wrong 1 in 3 times is dangerous in a way raw accuracy never shows.

Calibration metrics that would catch this — ECE, adaptive ECE, MCE, Brier score, NLL — exist in ML research but almost never show up in a reusable, submit-your-model-and-get-scored **platform**. Every team that wants this either skips it, or reinvents a one-off calibration script from scratch.
There is no honest, reusable, calibration-grade benchmarking harness a third party can point their own model at and get back the same rigorous report an internal team would get.

2. What Umpirix's Benchmark Submissions feature actually does

Umpirix already runs this evaluation internally, on its own models, on a public `/leaderboard`. **Benchmark Submissions** opens that same harness to anyone:

1. Go to `/submit-model`, give your model a label.
2. Point it at a live HTTP endpoint (Umpirix POSTs `{state, questions}`, your endpoint answers), **or** paste a precomputed answers JSON if you'd rather not stand up a live endpoint.
3. Umpirix runs the full 93-case gold suite through it and returns, in the same request: **accuracy, strict pass rate, ECE, adaptive ECE, MCE, Brier score, NLL** — overall and broken down per question type (choice / noul / score) — plus a real **execution trace** (steps, outbound/tool-call count, latency p50/p95, tokens + cost when reported, error/timeout counts).
4. Opt into a **fault-injection** stress test and Umpirix re-runs the whole suite with deterministically injected timeouts/5xx/malformed responses, reporting **robustness**: accuracy under faults vs. the clean run.
5. The submission appears live as a "Community submitted" row on the public `/leaderboard`, in the exact same table, same columns, as Umpirix's own models — not a separate, lesser page.

This is a first-class part of the real platform — a database-backed subsystem with auth, quota, and a public feed — not a CLI script or a demo page bolted on for the hackathon.

3. Real technical architecture (data + execution tracing)

- **Migration**: `migrations/0011_external_benchmarks.sql` — two tables, `benchmark_submissions` (submitter, model label, source: endpoint or answers upload, status) and `benchmark_results` (one row overall, one row per question type: accuracy, strict pass, ECE, adaptive ECE, MCE, Brier, NLL). `migrations/0012_benchmark_execution_trace.sql` adds two more additive, nullable columns for the execution trace + fault-injection result (no backfill, no breaking change to existing rows).
- **Execution trace + fault injection**: `instrumentRunner` (same file) wraps every submission's runner — steps, outbound/tool-call count, wall-clock latency p50/p95, tokens + cost when the submitter's own endpoint reports them, error/timeout counts. An opt-in, mulberry32-seeded fault-injection wrapper re-runs the suite with injected timeouts/5xx/malformed responses at configurable rates and reports robustness (faulted accuracy / clean accuracy) — real numbers in slide 7.

4. Real technical architecture (scoring, auth, SSRF, UI, API)

- **One scoring engine, not two**: `src/lib/jev/benchmark-scoring.ts` builds an HTTP-endpoint runner or a precomputed-answers runner, then feeds both into the *exact same* `evalGoldWith` (`src/lib/jev/gold.ts`) and `buildReliability` (`src/lib/jev/calibration.ts`) that score Umpirix's own internal models. A third-party submission and `system-one`'s own leaderboard entry are graded by identical code — no divergent "benchmark mode" math that could quietly drift from the real thing.
- **Auth + quota, not an open hole**: `src/lib/jev/benchmark-submissions.server.ts` wraps the scoring engine with the platform's real `resolveConsoleUser` auth and real `checkQuota` rate-limit/quota system — the same tables (`key_quotas`, `usage_events`) every other `/api/v1/*` route uses.
- **SSRF-guarded outbound calls**: `src/lib/jev/outbound-guard.server.ts`, a shared guard (extracted from the pre-existing outbound-webhooks protection, so there's one implementation instead of two that could drift) — rejects loopback / RFC1918-private / link-local / cloud-metadata / CGNAT targets, by both a literal-IP check and a real DNS-resolution check, before any submitted endpoint URL is ever called.
- **Real UI + API, in the real product**: `src/routes/leaderboard.tsx` renders completed submissions as extra rows in the same gold-suite table; `src/routes/submit-model.tsx` is the real submission form, linked from the leaderboard header. Backed by `src/routes/api/v1/benchmark-submissions.ts` (auth + quota enforced, `GET` / `POST`) and `.../benchmark-submissions.leaderboard.ts` (public, unauthenticated feed).

5. How Bifrost powers Umpirix

Umpirix is built **using Bifrost** (Maxim AI's open-source LLM gateway, `github.com/maximhq/bifrost`) — not a track name added after the fact. One env switch, `BENCH_LLM_GATEWAY=bifrost`, routes every LLM-backed call this platform makes — the leaderboard's `llm` row, the `auto` escalation cascade, and the judge stage of the router→gate→judge agent loop (`src/lib/jev/loop.ts`) — through Bifrost's universal OpenAI-compatible `/v1/chat/completions` instead of the provider directly, all through one choke point: `callBifrostStructured` in `src/lib/jev/llm.server.ts`.

Flow: Umpirix (agent-under-test / judge) → Bifrost gateway (`:8099`, `npx -y @maximhq/bifrost`) → real provider (anthropic / groq / openai / xai / ollama — Bifrost's provider slugs match this app's own 1:1) → real per-call telemetry read back from Bifrost's own response (`extra_fields`), not recomputed.

Real number (verified live, 2026-11-11)	Value
Gold-suite run through Bifrost → Anthropic claude-opus-5	93/93 real calls, accuracy **0.9794**
Bifrost's own measured latency (`extra_fields`, not our timer)	p50 **1567ms** / p95 **2324ms**
Real tokens (Bifrost-reported = this app's usage sum)	**106,543 in / 7,350 out**
Fault-injected re-run (15/15/10% timeout/5xx/malformed)	**67/93** real calls, robustness **0.621**
Real Bifrost **fallback** (primary model deliberately invalid)	5/5 served by fallback, `is_fallback: true`
Committed `/leaderboard` row, real **Gateway** badge	242 real calls, 273,825 in / 22,321 out tokens

6. Real proof it works (accuracy + calibration)

This was run end to end against a real local server, not described from source and not simulated:

A 93-case answers payload — built from Umpirix's own `system-one` engine's real answers — was submitted through the actual `/submit-model` browser UI (filled the label field, pasted the JSON, clicked "Submit and score," in a real headless-browser session). It returned:

```
ACCURACY 85.6%   STRICT PASS 100.0%   ECE 0.096   ADAPTIVE ECE 0.081
MCE 0.806        BRIER 0.127          NLL 0.828   CASES 93
choice n=48 acc=87.5% ece=0.136 | noul n=32 acc=78.1% ece=0.124 | score n=17 acc=94.1% ece=0.025
```

These numbers are **byte-identical** to `evalGold("system-one")`'s own internal report — proof the external-submission path is scored by the real engine, not a second, possibly-looser one built for a demo. It then showed up live as a "Community submitted" row on `/leaderboard`, confirmed with a post-hydration screenshot (not just the server-rendered shell).

SSRF guard exercised for real too: a loopback URL and the cloud-metadata address (`169.254.169.254`) both got real 400 rejections before any outbound request was attempted.

```
npm run typecheck && npm test   # clean; 138 passed, 1 pre-existing skip, 0 failed
npm run gold                     # 61 passed, 0 failed (unchanged by this work)
```

7. Real proof it works (execution trace + fault injection)

Also real, also run twice: a real HTTP POST to a running dev server (fault injection on, seed 42, 10% each of timeout/5xx/malformed) returned clean accuracy 0.8557 with a real-but-trivial trace (93 cases, 0 outbound calls — an "answers" replay makes no network call) and a faulted-run accuracy of 0.0722 (30 of 93 cases hit, robustness 0.0844) — then read back, trace and all, from the public `/api/v1/benchmark-submissions/leaderboard` feed.

Separately, `endpointRunner` against a REAL local HTTP agent (reporting real `usage` / `cost_usd` on its own response) made 93 real round trips: latency p50/p95 3.3ms/21.5ms, tokens in/out 3906/651, cost $0.011439 — all exactly 93x the per-call numbers the agent actually reported. Injecting faults against that same real endpoint (60% total fault rate) made only 58 of 93 real calls — proof `timeout` / `server_error` faults short-circuit before the real network call, not after.

Execution trace + fault injection (2026-11-10), also real, also run twice: the SAME machinery (slide 3) now also verified end to end with real Bifrost-routed calls (slide 5) — one tracing/fault-injection engine, proven against both a local answers replay and a live Bifrost→Anthropic run.

8. Honest scope and limitations

- **Self-host only.** There is no hosted `openjev.com` a judge can submit a model to right now — clone the repo and run `npm run dev` (or Docker) to try it. This deck does not claim a public SaaS instance exists.
- **The gold set is Umpirix's own.** 93 customer-support/agent-style choice/score/noul cases, including out-of-scope probes. This is a real, working calibration-grade harness for typed-decision models — not an independently-curated, cross-domain benchmark suite covering every possible agent task.
- **Synchronous, not queued.** Submissions run inline in the request (no background job queue exists in this app yet) — an endpoint submission is up to 93 sequential HTTP calls to the submitter's server.
- **Bifrost scope**: verified live with one real provider (Anthropic) via a file-based `config.json`, no dashboard admin account — Bifrost's `GET /api/logs` history and budget/rate-limit governance need `BIFROST_SETUP_TOKEN` + a login this pass does not provision, so telemetry comes from the inference response's own `extra_fields` instead. The fallback demo (slide 5) used one provider with a deliberately-invalid model as primary, not two real providers — real Bifrost-side failover, just not shown cross-provider.
- **SSRF guard is submission-time, not continuous.** It validates the URL before calling it; it does not defend against an endpoint compromised *after* that check.

9. Explanation of how AI is used in the solution

Two distinct, honest roles — neither is "call an LLM and hope":

1. **The thing being benchmarked is AI.** Every submission scores a *typed-decision model* — the kind of judgment an AI agent makes between tool calls (classify, route, gate, score-with-confidence). Umpirix's own leaderboard models (`system-one`, lexical baselines) are exactly this shape, and so is anything a third party submits.
2. **The benchmarking math itself is calibration science, not an LLM judge.** Accuracy/strict-pass/ECE/adaptive-ECE/MCE/Brier/NLL are computed deterministically from the submitted model's own stated confidence vs. the gold answer (`src/lib/jev/calibration.ts`) — no LLM-as-judge step that could itself be miscalibrated or inconsistent between runs.
3. **Where Umpirix calls a real LLM**: the typed-decision engine's escalation/`auto`/agent-loop-judge path can be backed by a hosted LLM provider, now **routed through Bifrost** (slide 5) as one of several interchangeable engines behind the same typed contract — a submitter can benchmark an LLM-backed agent exactly the same way as a pure classifier.

10. Why this matters, and what's next

Model benchmarking has converged on "did it get the right answer." Agent benchmarking needs a second axis: **can you trust the confidence it reported?** Benchmark Submissions makes that a one-form, no-setup check, reusing a production calibration engine instead of a one-off script — now with Bifrost as the real gateway every LLM-backed call runs through.

Shipped for real: execution trace (steps, latency p50/p95, tokens, cost, errors) and deterministic fault injection reporting robustness, live as real leaderboard columns and a per-submission trace view, verified against both local replays and a live Bifrost-routed run (slide 5).

Roadmap past the hackathon (honest, not promised — none of this is built):

- A hosted, public Umpirix instance, so a judge can submit without cloning.
- Bifrost dashboard-auth provisioning (`BIFROST_SETUP_TOKEN`), unlocking `/api/logs` history and budget/rate-limit governance as a second telemetry source alongside the inference response itself.
- A real multi-step agent runner, and named fault kinds for "unavailable model" / "misleading instructions" (today approximated by `server_error` / `malformed`).

11. Judging-criteria map

Criterion	Where it's covered
Innovation	Slide 1–2, 5: calibration (not just accuracy) as a public, submit-and-score platform feature, with real LLM calls routed through Bifrost
Technical execution	Slide 3–5: real migration + one shared scoring engine + auth/quota + SSRF guard + Bifrost gateway integration, verified end-to-end
Problem & potential impact	Slide 1, 6–7, 10: miscalibrated-confidence risk in deployed agents; execution-trace + fault-injection + Bifrost fallback scoring built and run with real numbers
Use of AI	Slide 9: what's benchmarked (typed-decision/agent models), how scoring stays deterministic, how Bifrost-routed LLM-backed agents plug in
User experience	Slide 2, 6–7: one form (`/submit-model`), same table as the native leaderboard, real trace + robustness + Gateway badge shown inline
Scalability & potential	Slide 10: pluggable gold sets, hosted instance, job queue, Bifrost governance provisioning, a real multi-step agent runner

12. Team

Built and maintained as the open-source decision substrate behind a small products portfolio (`github.com/caelum0x`) — this hackathon entry is a real feature of a real, already-running platform, not a team assembled for the event or a repo created for the submission. The Benchmark Submissions subsystem itself (migration 0011, `benchmark-scoring.ts` , `benchmark-submissions.server.ts` , `outbound-guard.server.ts` , `submit-model.tsx`) was built in commit `c70f3d0` on 2026-11-07, inside the hackathon's official Nov 1–15 build window; the surrounding platform (leaderboard, calibration engine, auth/quota system) is the pre-existing infrastructure Rule 1 explicitly allows reusing. The execution-trace + fault-injection half (migration 0012) landed 2026-11-10, and the Bifrost gateway integration (slide 5) landed 2026-11-11, both inside the same build window.

13. Links

Repo: `github.com/caelum0x/openjev` (package/repo name unchanged; project title is Umpirix) · **Try it live (self-hosted):** `/leaderboard` and `/submit-model` after `npm run dev` · **Full writeup:** `hackathons/geekroom-2026/HACKATHON-SUBMISSION.md` · **Bifrost run output:** `hackathons/geekroom-2026/bifrost-run-results.json` · **Demo video:** `hackathons/geekroom-2026/demo.mp4`