Chalk

Building a controlled AI evaluation system

Applied AI | Python + SQLite | local + cloud inference | Linux automation

What Chalk is

A private system for testing how consistently local and cloud language models make structured probabilistic forecasts. NFL games provide a controlled domain: scheduled events, repeated conditions, and objective outcomes.

The question

What happens when multiple local and cloud language models are asked to make the same probabilistic forecasts under controlled conditions - with bounded inputs, predeclared work, preserved failures, and repeatable evaluation?

The experiment

Each model receives the same deliberately limited packet. One paired prompt adds explicit calibration instructions; market odds are withheld from model inputs and stored separately for later comparison. Every forecast is traced to its inputs, model profile, prompt, timing, attempt history, and acceptance state.

CURRENT REGULAR-SEASON SNAPSHOT

31	666 / 744	4 x 2 x 3
regular-season events	accepted / expected forecast cells	model profiles x prompts x samples

System-wide engineering check: 559 tests passing in the full current suite.

Current experiment design

- 4 model profiles spanning cloud APIs and local Ollama inference
- 2 paired prompt variants using the same bounded input packet
- 3 samples per arm to observe agreement and instability
- 24 predeclared cells per game so missing work remains visible

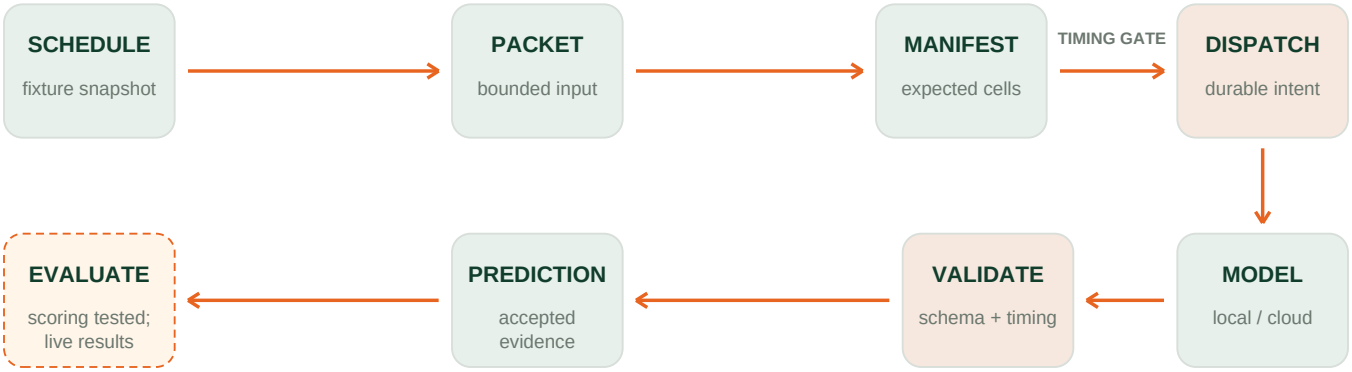
CURRENT BOUNDARY

Forecast capture is operational. Scoring code and tests exist, but live result ingestion is not yet populated; there are no finished current-season scoring results.

Not a betting product.
Football is the experimental instrument: structured events, repeated observations, objective outcomes.

Architecture: an evidence pipeline

Chalk is designed around the lifecycle of an observation, not around a single model call.



Predeclared work

Every expected cell remains in capture reporting, including failures. Forecast-quality comparisons use matched completed cases.

Attempts are not predictions

Dispatch, attempt, and accepted forecast are separate states. A failed API call cannot masquerade as a missing row.

Local runtime identity is checked

Official local runs can be gated against a frozen model/runtime fingerprint rather than trusting a model name.

Failures stay visible

Technical, schema, timing, and not-attempted states are preserved instead of being silently retried into a cleaner history.

REAL INCIDENT / SEPT. 19

30 expected local-model cells failed. I left them failed.

A Windows/Ollama runtime outage exposed a dependency on my local process management across five games. Chalk recorded 30 expected cells as technical failures before inference dispatch. After restoring the runtime, verifying its fingerprint, and restoring the pre-recovery database state, I retained the failures rather than backfilling forecasts.

runtime unavailable | 30 failures recorded | runtime restored | fingerprint verified | pre-recovery state retained

What this project demonstrates

The strongest evidence is not that a model can pick football games. It is the process of turning a vague question into an operating, inspectable system.

Systems thinking

Separated schedule, packet, prompt, profile, manifest, dispatch, attempt, prediction, benchmark, result, and evaluation into explicit layers.

Reliability thinking

Built around idempotency, durable uncertainty, terminal states, runtime identity checks, and preserved evidence; the Sept. 19 outage exercised those boundaries in a real failure.

Experimental design

Paired prompts use the same limited packet; market odds are withheld; three samples are aggregated within an event/configuration rather than treated as independent outcomes.

Technical operation

Python + SQLite, model APIs, local Ollama inference, Linux/systemd, SQL inspection, telemetry, Git checkpoints, and home-lab operations.

Relational data

A normalized schema keeps expected work, attempts, predictions, usage, telemetry, and results at their own grains so failures and missingness stay interpretable.

AI-assisted development

AI has contributed heavily to code, tests, debugging, and documentation. I retain responsibility for experimental design, operating decisions, validation, and evidence integrity.

My role

I am the system owner/operator using substantial AI-assisted development. I define the experiment and controls, choose and freeze model/prompt profiles, operate the system, investigate failures, validate behavior, and decide what counts as official evidence.

CURRENT LIMITS

- Not enterprise production software or an enterprise AI deployment.
- Current official sample is still small; no model is demonstrated to be "best."
- Scoring code and tests exist, but live result ingestion is not yet populated, so current-season scoring is not a finished public result.

WHY CHALK MATTERS

The project began as a simple question about LLM forecasts. It became an unattended instrument that predeclares work, executes across local and cloud models, detects failure, preserves evidence, and is built to support later evaluation. That evolution is the point: imagination chose the question; ground truth shaped the system.