Anton Nesterov · an@vski.sh · August 23, 2026

The VSKI Whitepaper: Auditable Intelligent Systems on a Stock-and-Flow Substrate

Systems · Neuro-Symbolic · Simulation

VSKI and VSKI Lab, formally. Conservation as the contract between neural and symbolic components, the platform that enforces it, and two worked use cases: a probability space as a flow network, and a delivery kitchen as queueing plus accounting.

doi: 10.5281/zenodo.22073020

Assets & Materials

VSKI (home page and documentation)	https://vski.ai
VSKI Lab (the app)	https://app.vski.ai
VSKI platform (source and wiki)	https://vski.sh/x/platform

Abstract

VSKI (the Vector Semantic Knowledge Index) is a self-hosted platform for building data-driven intelligent systems, and VSKI Lab is its modeling environment. Together they implement one idea: **the symbolic layer holds the truth and asserts it, while the neural layer proposes and is verified**. The core formal object is a directed graph of stocks and flows in which conservation of quantities is asserted on every tick, so that accounting errors, probability mass that appears from nowhere, or money that vanishes are rejected by construction rather than discovered after the fact. The platform provides durable storage (typed collections, SQL views, full-text and vector search), event-sourced workflows, and a records API; the Lab provides the modeling loop: compose the system, run it tick by tick, fence and audit agent interventions, capture trajectories, declare objectives, and train replacements inside the same harness. This whitepaper states the design principles, formalizes the model, and demonstrates the framework on two use cases: an academic one (a probability space as a flow network, where flow conservation *is* the law of total probability) and a business one (a delivery kitchen, where conservation *is* double-entry bookkeeping and the interesting behaviour is queueing).

1. Introduction

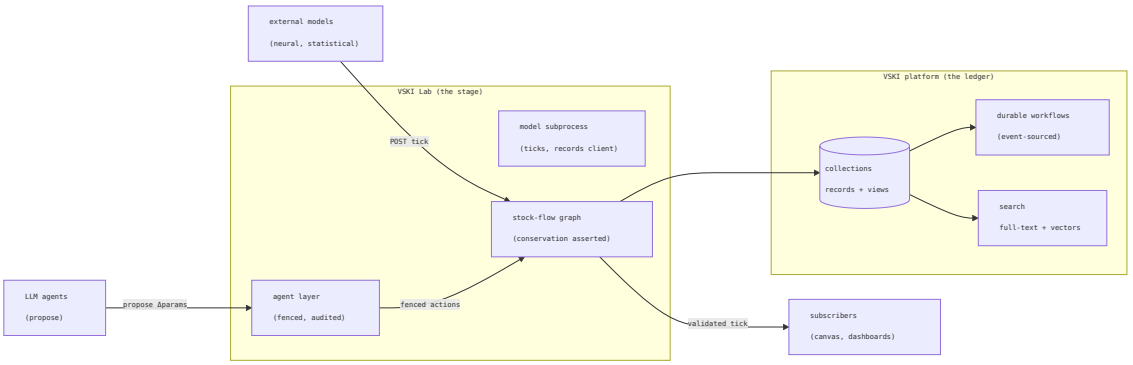
Intelligent systems are currently assembled from disconnected parts: a managed database, a workflow SaaS, a notebook environment, and a large language model (LLM) API stapled on top. The data leaves the walls, the logic disappears into configuration dialogs, and when the auditor asks why the system did what it did, the honest answer is a shrug at a token stream.

The problem is sharper than tooling. LLMs are the most capable perception-and-proposal engines ever built, and they are structurally unreliable as systems of record. They state wrong numbers with the same confidence as right ones; they do not conserve anything (ask an LLM to simulate a supply chain and the money quietly stops adding up); running them over every tick of a live system is a per-token meter doing arithmetic a spreadsheet does for free; and their reasoning is not something you can file.

The classical modeling tradition has the mirror-image strength. Stock-and-flow models have been used to run economies, epidemics, and factories for a century: Forrester’s industrial dynamics [1] formalized the feedback structure of enterprises; the Kermack-McKendrick epidemic model [2] is three compartments with people flowing between them. These models are precise, cheap, deterministic, and auditable. What they cannot do is meet the world on its own terms: read a document, parse a complaint, or judge which of ten conflicting signals matters.

VSKI is built on the position that the two should be composed under a specific contract, on infrastructure you own. This whitepaper makes four contributions:

1. **A contract** (Section 2): conservation is the interface between the neural and symbolic components. The neural side proposes; the symbolic side enforces invariants and computes exact consequences. A proposal that breaks accounting is rejected before anyone sees it.
2. **A platform** (Section 3): a single self-hosted service providing typed collections, a records API, views, full-text and vector search, realtime, and event-sourced durable workflows. Every dataset, recorded run, and telemetry stream in the system is a collection.
3. **A formalism** (Section 4): the stock-flow graph with signal edges, live parameters, fenced agents, declared objectives, and a capture-train-deploy loop, stated precisely.
4. **Two worked use cases** (Sections 5 and 6): an academic one in which the probability axioms are enforced as flow conservation, and a business one in which queueing theory and double-entry bookkeeping are live and interactive.



2. The contract: propose and verify

Kautz's taxonomy of neuro-symbolic integration [3] spans six categories, from tight symbolic-neural embeddings to loose cooperation through a narrow interface. VSKI sits deliberately at the loose end, the category Kautz calls *Neuro | Symbolic*: two systems with different native strengths, cooperating through a contract. The same loose pattern underlies DeepMind's AlphaGeometry, in which a neural proposer and a symbolic deduction engine cooperate to solve olympiad geometry at near gold-medal level; in the published ablations, neither half alone comes close [4].

The contract has three clauses:

P1. The symbolic layer owns the truth. State lives in a stock-flow graph whose accounting is asserted, not trusted. Any component (an LLM, a learned policy, an external neural model) interacts with the state only by proposing changes through a narrow interface. The symbolic layer checks the proposal, applies it, computes the exact consequences, and logs it.

P2. The neural layer owns the judgment. Reading unstructured input, ranking conflicting signals, proposing interventions: these are learned competences, and the framework places no constraint on how they are implemented (an API call, a torch checkpoint, a rule-based

heuristic). The constraint is on scope, not method: a fence declares which parameters an agent may touch, which events it may emit, and which workflows it may start.

P3. Everything is replayable and inspectable. Every tick, every agent decision with its rationale, every recorded run, and every workflow step is a row in a collection on the platform. The audit trail is the substrate, not an afterthought.

This division is what makes the neural side safe to iterate on: the fence and the invariant checker do not change when the model behind the proposal changes from a heuristic to a trained policy.

3. The platform: one substrate for data and automation

VSKI's data backbone is a single self-hosted service. Its storage unit is the **collection**: a typed record store defined at runtime (field types text, number, bool, json; optional required, unique, default), with records carrying `id`, `created`, and `updated`. Access is a records API (create, paged list with a filter language and sort, partial update, delete), plus full-text search and vector embeddings over the same records. Collections are scoped by database; the Lab provisions one database per project.

Views make aggregation declarative. A view is a SQL `SELECT` (GROUP BY included) stored in the collection's options and materialized by the platform. Reports read views; no client ever computes SUM on the fly. The aggregation being configuration rather than code is what keeps historical reporting audit-grade: the definition is versioned with the collection.

Durable workflows give the pattern long-horizon reach. A workflow is Python code over idempotent steps, event-sourced on the platform in the saga tradition [5]: runs survive process restarts, suspend mid-run on signals (a human approval gate is one `await`), and resume exactly where they stopped. This is the execution substrate for everything an intelligent system must do across hours or days: batch retraining, settlement, notification chains.

Retention is default-on housekeeping. Log collections grow without bound in most stacks; here a background pass applies each project's declared policy, pruning processed events and stale telemetry oldest-first to row caps (recorded datasets are training data, not logs, and are kept unless capped explicitly). Long runs do not slowly fill the disk, and the pruning is records-API deletion, never raw SQL against storage.

The sovereignty claim is literal: the platform is one binary (or small Docker stack) on your hardware, with no outbound dependency. Data, models, and logs never leave the machine by default.

4. VSKI Lab: the formalism

4.1 Stocks, flows, conservation

A model is a directed graph $G = (V, E)$. Each node $v \in V$ holds a stock $x_v(t) \in \mathbb{R}_{\geq 0}$; each edge $e = (u, v) \in E$ carries a flow $f_e(t) \geq 0$, an amount per tick. Nodes are partitioned into conserving nodes C (interior nodes with one unit: orders, meals, euros, probabil-

ity mass) and boundary nodes B (deliberately mixing units, e.g. goods priced into money). For every conserving node, on every tick:

$$x_v(t+1) \;=\; x_v(t) \;+\sum_{e=(\cdot,\,v)} f_e(t) \;-\sum_{e=(v,\,\cdot)} f_e(t), \qquad v \in C$$

The engine asserts this identity within a tolerance $\varepsilon = 0.01$. A tick that violates it is rejected with an HTTP 422 before any subscriber sees it. This is the load-bearing design decision of the whole framework: conservation is not a test the developer writes; it is a property the substrate enforces on every path, including ticks computed by entirely external neural models and parameter changes proposed by agents.

Bottlenecks fall out of the same formalism. Giving edges capacities makes G a flow network, and the max-flow min-cut theorem [6] identifies the exact set of edges that caps throughput. A growing stock at a node is not a metaphor for congestion; it is congestion, with a number on it.

4.2 Signals break cycles

Real systems have feedback: queues back up, servers throttle, queues drain. A naive graph encodes this as a cycle, which no topological evaluation can handle. VSKI partitions edges into resource edges (carrying conserved stuff) and **signal edges** $S \subseteq E$ (carrying read-only information, no mass). A signal's value is one tick old:

$$f_s(t) \;=\; \phi_s\big(x(t-1)\big), \qquad s \in S$$

so the resource subgraph $(V, E \setminus S)$ remains a directed acyclic graph and evaluates in topological order, while feedback resolves over time exactly as it does in the world. This is the standard system-dynamics move [1], made structural: the acyclicity is a property of the edge types, not of careful manual layout.

4.3 Parameters are live data

Every knob (prices, capacities, rates, and in Section 5, probabilities) is a row in a platform collection, re-read by the model on every tick:

$$p(t) \;=\; \mathrm{records}[\mathrm{params}](t)$$

Editing a cell in the grid is a live configuration change effective on the next tick, with no redeploy. This unifies operator input, agent action, and calibration: all three write the same rows through the same API, and all three are logged identically.

External input rides the same discipline. Events (shocks, bursts, agent emissions) land in a queue that the model drains each tick, and an event may carry an explicit due tick, so a consequence scheduled for fifty ticks hence is a first-class row in the queue rather than a sleeper loop in model code.

4.4 Agents and the capability fence

An agent is a policy π that observes the posted state and proposes parameter deltas Δp; formally the runner applies

$$p(t+1) \;=\; p(t) \;+\; \Delta p, \qquad \Delta p \in F_\pi(p)$$

where $F_\pi(p)$ is the capability fence: the set of parameter writes the configuration grants (an empty fence means read-only). The fence bounds magnitude as well as scope: every granted write is clamped to the parameter's declared range and, when the agent declares it, to a maximum per-tick delta, and each clamp is logged as a decision annotation. A run-level kill-switch disables agent application entirely, passing ticks through unmodified, so a misbehaving policy can be silenced without stopping the system it sits on. Agents may additionally emit events into the model's queue and start workflows, each grant named explicitly. Every intervention lands in the decisions feed with its rationale, attached to the node it touched. The next tick then shows the consequence of the intervention on the canvas: cause, effect, and record in one place.

4.5 Objectives, capture, and the training loop

The framework treats evaluation as a first-class declaration. An `objectives.json` scores every captured trajectory

$$\tau = (s_0, a_0, s_1, a_1, \ldots, s_T)$$

against named criteria (minimize blackout energy, maximize cash, under hard constraints), and a recording mode persists every tick (state, parameters, agent deltas, agent actions) as a durable, seeded, reproducible dataset. The loop is then closed inside one environment [7]:

1. **Observe** the baseline (a rule-based heuristic is the number to beat).
2. **Capture** trajectories from live runs.
3. **Train** a replacement policy offline against the declared objectives.
4. **Deploy** by swapping an artifact pointer; the fence, the audit trail, and the objectives apply to the trained policy unchanged.

Because the validation envelope is external to the policy, a heuristic, a torch checkpoint, and an LLM are interchangeable in the same slot, and each is measurable in the same harness.

5. Use case I (academic): a probability space as a flow network

Probability theory is taught with Venn diagrams, which are unreadable past three variables. The VSKI Lab *Probability Lab* rebuilds the sample space as a flow network, and the mapping makes the axioms structural rather than notational. Kolmogorov's axioms [8] state that probabilities are non-negative, that $P(\Omega) = 1$, and that probabilities of disjoint events add. In the flow encoding:

- **Normalization.** A source node Ω emits exactly 1.0 per tick. The axiom is a rate knob; dropping it below one visibly breaks the space at the sink.
- **Additivity / the law of total probability.** Conservation at a node *is* the partition law. At node A, with mass routed to the joint regions by the conditional $P(B \mid A)$:

$$\underbrace{f_{\Omega \to A}}_{P(A)} = \underbrace{f_{A \to A \cap B}}_{P(A \cap B)} + \underbrace{f_{A \to A \cap \neg B}}_{P(A \cap \neg B)}$$

- **Conditional probability as routing.** Edge flows factor as $f(u, v) = f(u)\, P(v \mid u)$: conditioning is literally where the mass goes.
- **Bayes' rule as a ratio of two edge flows.** With $P(A) = 0.3$, $P(B \mid A) = 0.8$, $P(B \mid \neg A) = 0.2$ (the shipped defaults):

$$P(A \mid B) \;=\; \frac{f(A \to A \cap B)}{f(A \to A \cap B) + f(\neg A \to \neg A \cap B)} \;=\; \frac{0.24}{0.24 + 0.14} \;\approx\; 0.632$$

The platform asserts the axioms on every posted tick: interior nodes hold zero stock, so the conservation check reduces to $\sum f_{\text{in}} = \sum f_{\text{out}}$ at each node and 1.0 at the sink. A model that mints or leaks probability mass is rejected with a 422 before any subscriber sees it. Only three numbers are free ($P(A)$, $P(B \mid A)$, $P(B \mid \neg A)$); complements are derived as $1 - x$, so an inconsistent space is not validated after the fact but *inexpressible*.

Two pedagogical payoffs. First, the **base-rate fallacy becomes visible**: the medical preset (prevalence 1%, sensitivity 95%, false-positive rate 5%) gives

$$P(\text{disease} \mid +) \;=\; \frac{0.01 \times 0.95}{0.01 \times 0.95 + 0.99 \times 0.05} \;\approx\; 0.161,$$

and the flow diagram shows why: the false-positive ribbon over the healthy 99% dwarfs the true-positive ribbon. Second, **the representation scales the way the theory does**: the same shift from sets to directed graphs underlies Bayesian networks [9] and the transition structure of Markov decision processes. A fifty-variable model is not a fifty-dimensional Venn diagram; it is more nodes, with conservation asserted at each.

Arranged in topological order, the edge flows are the nonzero cells of an upper-triangular adjacency matrix; row sums equal column sums at every intermediate node, which is the law of total probability in matrix form. The Lab renders this live as a heatmap and as a Sankey flow, both reading the same model object the notebook reads, so the three views cannot drift.

6. Use case II (business): the ghost kitchen

The second use case is a delivery-only restaurant: demand arrives stochastically, capacity is finite, money flows one way and goods the other. It is deliberately mundane, because the framework's business claim is that operational economics is exactly where conservation pays.

Demand is a Poisson process with a rush cycle:

$$N(t) \sim \text{Poisson}\big(\lambda(t)\big), \qquad \lambda(t) \;=\; \bar{\lambda}\,\big(1 + \alpha\, r(t)\big)$$

where $r(t)$ is a periodic rush shape (lunch and dinner) and α is a live knob (rush amplitude, shipped at 50%). Orders queue at the kitchen, which serves at capacity μ per tick; with the shipped defaults $\bar{\lambda} \approx 10$ and $\mu = 12$, the utilization is

$$\rho \;=\; \frac{\bar{\lambda}}{\mu} \;\approx\; 0.83 \;<\; 1$$

so the system is stable in the queueing-theoretic sense [10]: rushes spike the backlog, calm hours drain it, and the long-run behaviour is governed by Little's law, $L = \lambda W$ [11], which the model exhibits rather than asserts (waiting riders, in-transit meals, and busy drivers are stocks you can point at). The pedagogical moment is live: drag the kitchen capacity below the arrival rate and ρ crosses 1, at which point the fluid dynamics of the backlog,

$$Q(t+1) \;=\; \big[\,Q(t) + \lambda(t) - \mu\,\big]^{+},$$

predict linear divergence, and the canvas shows exactly that: the backlog node grows without bound until capacity is restored. Queueing theory as a thing you watch instead of a formula you trust.

The money side is double-entry bookkeeping, enforced. Pacioli's 1494 codification [12] made every debit have a credit; here the conservation check plays the auditor. With the shipped parameters (menu price p = \12, $gatewayfee$\gamma = 3%, $ingredientcost$c = \$3.50$permeal$, $wages$w = \$18\$ per tick, plus a per-order courier fee), the cash account evolves as

$$C(t+1) \;=\; C(t) \;+\; \underbrace{(1-\gamma)\,p\,s(t)}_{\text{net revenue}} \;-\; \underbrace{c\,s(t)}_{\text{ingredients}} \;-\; \underbrace{w + \kappa\,s(t)}_{\text{wages + courier}},$$

where $s(t)$ is meals served. At the defaults the seeded economy accrues roughly +\23\$ per tick. An economics bug (a meal priced into the wrong account, a fee that vanishes) is a rejected tick, not a silent drift. Two participants are deliberately external services (courier fleet, payment gateway): a realistic business model has pieces you do not control, only pay, and the graph keeps their fees honest.

Every generated demand arrival is logged as an event; every tick appends a telemetry row; historical dashboards read SQL views aggregated by the platform over that record. The same recording machinery from Section 4.5 turns the kitchen into a training environment: the knobs an agent may turn (the fence), the objectives ($P\&L$ under a service-level constraint), and the captured trajectories are the ingredients for a learned pricing or staffing policy that deploys into the identical scaffold.

7. Discussion and related work

System dynamics. The stock-flow formalism descends directly from Forrester [1] and the simulation tradition he founded; the differences are the substrate (a durable, searchable, self-hosted platform instead of a simulation file), the enforcement (conservation asserted at the boundary on every externally computed tick), and the agent layer with its fence.

Neuro-symbolic integration. The propose-and-verify contract is the loose-coupling end of Kautz's taxonomy [3]; AlphaGeometry [4] is the flagship evidence that the pattern buys capability neither half has alone. VSKI's contribution is applying the same contract to *operations*: the symbolic half is a running business or physical system, and the neural half is judged by declared objectives on captured trajectories rather than by benchmark accuracy.

Probabilistic graphical models. Section 5's probability-as-flow representation is a didactic cousin of Bayesian networks [9]: both replace set-based sample spaces with directed structure over which local conditionals compose into global consistency. The flow encoding adds something the diagram does not have: a runtime that rejects inconsistent mass routing.

Queueing and accounting. The business use case is deliberately textbook: Poisson arrivals, utilization, Little's law [10, 11], and double-entry [12]. The claim is not novelty in the theory but in the *liveness*: the textbook quantities are stocks and flows under live knobs, so every classical result is reproducible by poking the running system.

Durable execution. The workflow engine follows the saga model [5] with event-sourced state, deterministic replay, and signal-based suspension, which is what lets an approval gate survive a restart.

Limitations. The framework advances in discrete ticks; events can be scheduled for future ticks, but sub-tick dynamics remain tick-resolution by design, so phenomena that live between ticks are approximated rather than resolved. Conservation is asserted per conserving node, which pushes unit-mixing to explicit boundary nodes, a modeling discipline rather than a formal guarantee at the boundary. The capability fence bounds the scope and the magnitude of an agent's actions, not their intent; an agent granted a knob can still turn it badly within bounds, which is why the decisions feed, the kill-switch, and the objectives harness exist. And the self-hosted stack is single-tenant by design.

8. Conclusion

VSKI and VSKI Lab implement a simple division of labor with a hard interface: neural components propose, symbolic structure disposes, and conservation is asserted on every tick by a substrate that also stores, indexes, and orchestrates. The two use cases show the range: the same invariant checker that enforces the probability axioms on an academic teaching model also enforces the accounting on a business model, and the same fence-and-capture machinery that audits a rule-based controller accepts a trained replacement without changing anything else. The framework is general over domains precisely because it commits to so little: a graph, a conservation law, a records API, and a contract.

References

[1] J. W. Forrester, *Industrial Dynamics*. Cambridge, MA: MIT Press, 1961.

[2] W. O. Kermack and A. G. McKendrick, “A contribution to the mathematical theory of epidemics,” *Proceedings of the Royal Society of London. Series A*, vol. 115, no. 772, pp. 700-721, 1927.

[3] H. A. Kautz, “The Third AI Summer: AAAI Robert S. Engelmore Memorial Lecture,” *AI Magazine*, vol. 43, no. 1, pp. 105-125, 2022. doi:10.1002/aaai.12036

[4] T. H. Trinh, Y. Wu, Q. V. Le, H. He, and T. Luong, “Solving olympiad geometry without human demonstrations,” *Nature*, vol. 625, pp. 476-482, 2024. doi:10.1038/s41586-023-06747-5

[5] H. Garcia-Molina and K. Salem, “Sagas,” in *Proc. 1987 ACM SIGMOD Int. Conf. on Management of Data*, San Francisco, CA, 1987, pp. 249-259. doi:10.1145/38713.38742

[6] L. R. Ford and D. R. Fulkerson, “Maximal flow through a network,” *Canadian Journal of Mathematics*, vol. 8, pp. 399-404, 1956.

[7] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. Cambridge, MA: MIT Press, 2018.

[8] A. N. Kolmogorov, *Grundbegriffe der Wahrscheinlichkeitsrechnung*. Berlin: Julius Springer, 1933; English translation: *Foundations of the Theory of Probability*. New York: Chelsea, 1950.

[9] J. Pearl, *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. San Mateo, CA: Morgan Kaufmann, 1988.

[10] L. Kleinrock, *Queueing Systems, Volume 1: Theory*. New York: Wiley, 1975.

[11] J. D. C. Little, “A proof for the queuing formula L = λW,” *Operations Research*, vol. 9, no. 3, pp. 383-387, 1961. doi:10.1287/opre.9.3.383

[12] L. Pacioli, *Summa de arithmetica, geometria, proportioni et proportionalita*. Venice, 1494.