The Inference Control Plane

Part One: The Placement Contract

Where inference runs
determines what it can know,
what it can do,
and whether it can be trusted.

Contents

OPERATIONAL AI · AN APPLIED REASONING ESSAY

The Inference Control Plane

Part One: The Placement Contract

Where inference runs determines what it can know, what it can do, and whether it can be trusted.

By Stephen Alexander

AUTHOR'S NOTE

The earlier essays left me with a loose end. *Operational AI Becomes a Distributed Systems Problem* described operational AI as a portfolio of bounded functions spread across a distributed system. *Operational AI in Search and Rescue* showed how connectivity, urgency, available compute, and data boundaries can force an inference task to run in a different physical location. *Evidence Debt* examined what must be preserved if the resulting decisions are to remain assessable.

Put together, those ideas raise a practical question: when circumstances force inference to run somewhere else, how do we know it can still make a valid judgment there?

I spend much of my time around public-sector missions and distributed cloud infrastructure, where placement is commonly discussed in terms of latency, capacity, availability, locality, and cost. Those conditions matter, but they do not tell us whether a particular judgment remains trustworthy after it moves. That also depends on the model's behavior in its deployed configuration, the context available there, what the output is allowed to cause, how the function behaves when conditions deteriorate, and what can be demonstrated afterward.

This essay is an attempt to bring those concerns into the placement decision itself.

It uses a hypothetical public-sector case involving customs officers and physical shipping containers at a national border. It is not a description of any country's current systems or procedures. The scenario is useful because it brings local observation, remote records, time pressure, unreliable connectivity, data boundaries, and reserved human authority into one setting.

The same questions could arise for a mobile public-health team working far from a well-connected clinic or for a technician diagnosing critical equipment at a water facility, railway, or other public asset. The operational details would change; the placement problem would remain.

These are my own views, not my employer's, and not a forecast from anyone's company.

> ***Note on method.*** *I used AI tools to test the structure, challenge assumptions, compare alternative framings, and refine the prose and figures. The sources and influences listed at the end helped ground the argument and check its claims. AI was a reasoning and production aid, not an authority. The argument, editorial choices, and responsibility for any errors are mine.*

THE ARGUMENT IN BRIEF

Interactive AI is paced by a person. Operational AI is increasingly paced by events. Once an inference function is embedded in an automated workflow, the volume of decisions can exceed anyone's ability to inspect each output before the system continues.

That changes what infrastructure has to know. A scheduler can report that the required compute was available, a service was healthy, and a request completed successfully. Those facts do not establish that the model remained within its decision-quality threshold, that its context was current, or that the output was used within its authorized scope. Those conditions have to be represented explicitly.

This essay introduces a placement contract for doing so. Each bounded inference function declares what a valid judgment requires. Each execution environment declares what it can provide and enable. Bringing them together produces an execution contract for that particular function–environment pairing.

A valid pairing must provide enough without allowing too much. It must sustain the required decision quality, context, operating behavior, and record production while keeping access and effects within their permitted limits. That finding also has a lifetime. Models, policies, data, load, connectivity, and integrations continue to change after deployment.

Regulation and governance practice are already increasing the demand for records that make consequential AI decisions assessable. The relevant evidence is the observable basis of the decision, not a model's private reasoning or a persuasive explanation produced afterward.

Maintaining valid pairings as those conditions change is the beginning of an inference control plane. The practical starting point is one consequential function: compare the environments in which it could run, record where the contract holds, and make the gaps explicit.

Placing an inference function is never only a decision about where it runs. Its execution environment constrains which model it can use and the data and tools it can reach. It also shapes response time, behavior during a network outage, the handling of sensitive data, operating cost, and the authority that surrounding systems grant its output. Deployment practice routinely accounts for conditions such as latency, cost, availability, locality, residency, and resource capacity. What it rarely represents is the full set of conditions under which a particular judgment remains valid.

Consider a cargo vessel arriving in port with hundreds of shipping containers. In this hypothetical, **Secondary Inspection Referral** is a bounded inference function that estimates whether a particular container should be sent for additional inspection. Keep its purpose and authority fixed and change only where the inference runs: on a device near the inspection lane, at a regional facility, or in a national data center. It is still answering the same question, but from a different set of conditions each time.

Reasoning about that difference requires a way to describe the conditions under which the function produces a valid judgment and to test whether a placement provides them. That description is the primitive from which an inference control plane can be built: a contract for each function–environment pairing and a process that keeps the pairing valid as conditions change.

Placement starts with a required contract, derived from operational, mission or business, and governance requirements. It defines the quality the judgment must sustain; the context the function must reach and the limits on what it may access; its latency and compute budgets; its behavior during degradation; the effects its output may trigger; and the records it must produce.

The operator of each execution environment maintains a capability declaration. It describes measured model behavior under applicable conditions, available models and compute, access to data and tools, connectivity, response characteristics, data-handling controls, record production, and the mechanisms wired to the function's output.

Reconciling the required contract with that declaration produces an execution contract for the pairing. One check catches an environment that provides too little; the other catches a pairing that allows too much.

> ***“A placement is valid only when the environment supplies everything the function requires without giving it access or enabling effects beyond what the required contract permits.”***

These are checks on the pairing, not a permanent certification of either side. An environment may offer capabilities the function is prohibited from using, and a function may require conditions the environment supplies only at certain times or under certain loads. The execution contract identifies the allowed intersection and makes its assumptions explicit.

Secondary Inspection Referral's required contract begins by defining a usable estimate. It sets decision-quality and calibration thresholds for the inspection data the function is expected to encounter. If the function cannot stay within those limits, it must abstain instead of returning an estimate that appears more reliable than it is.

The contract also defines the information and time available. Secondary Inspection Referral must reach at least the shipment manifest and the importer's history, and it must return before the inspection workflow must decide whether the container continues through ordinary processing or enters the secondary-inspection queue. Its output may refer a container to that queue, but it may not place a hold, authorize the container to be opened, or authorize its seizure. Those decisions remain with an authorized officer. A defined fallback applies when normal operating conditions fail.

Finally, the contract states what must be recorded. For each estimate, the record preserves the context and source versions used, relevant tool and policy results, model and prompt versions, output, and uncertainty. Any explanation generated by the model is identified as model-provided rather than treated as independent evidence.

Workflow determines when the function executes; placement determines where. Here the workflow never changes: Secondary Inspection Referral runs when a container arrives and a referral decision is due. Only the execution environment varies.

The contract raises two questions that are easy to collapse. The first is where a fixed function can validly run. Figure 1A compares a device with a mobile node while holding the model, inputs, output, and authority constant. Both environments can perform the inference. The device preserves autonomy at the point of action but cannot produce all the evidence the contract requires. The mobile node produces the required basis but depends on the device-to-node link. “Sufficient” and “insufficient” are findings against the declared evidence obligation, not judgments made by the record itself.

FIGURE 1A

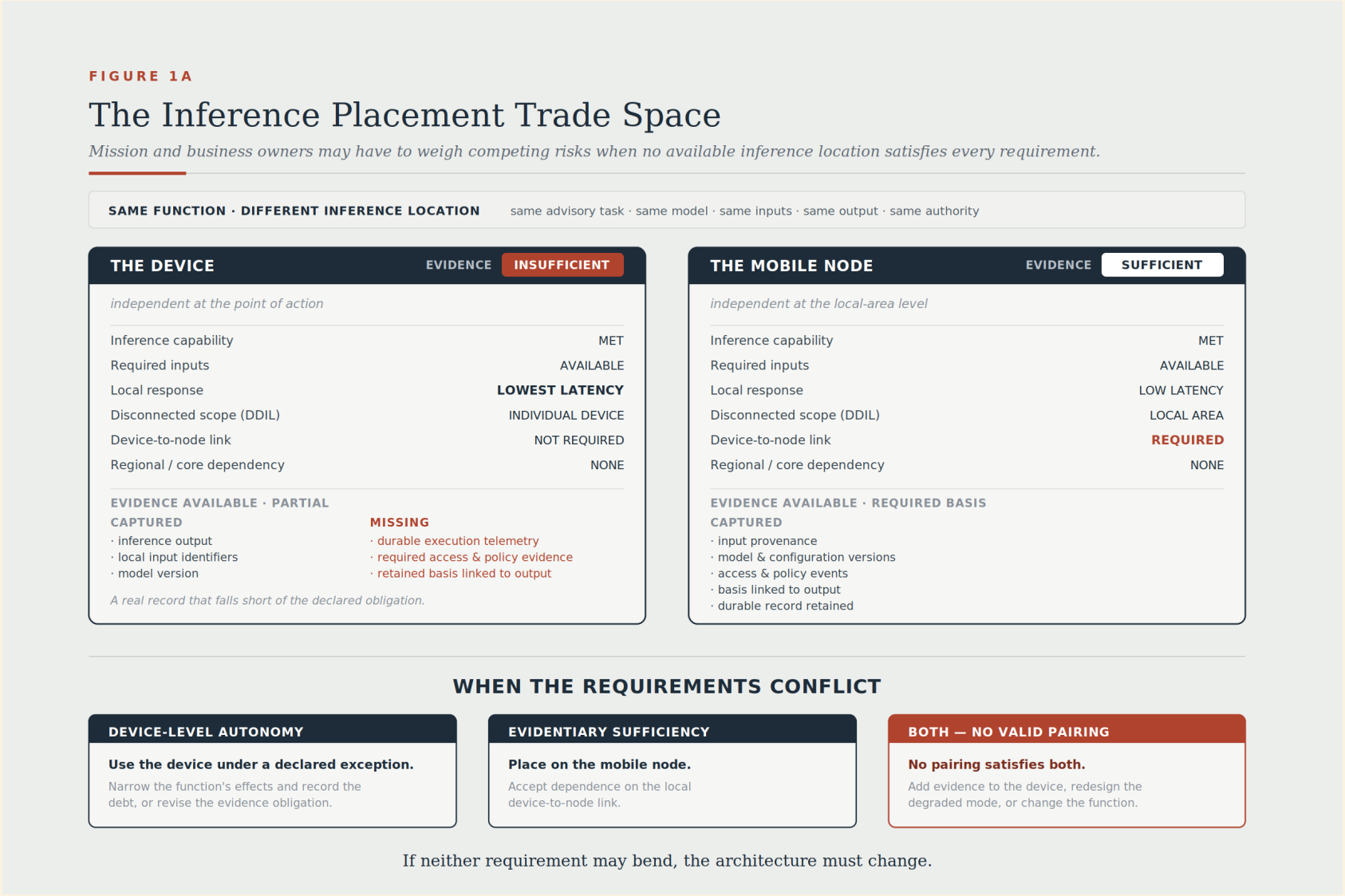

Figure 1A. *Neither environment is inherently better. If both device-level autonomy and the existing evidence obligation are non-negotiable, neither pairing is valid; the architecture or the function must change.*

The second question is what remains fixed when model weights move. Figure 1B follows one approved model into a regional synthesis and coordination function. The weights contribute the inference, but context assembly determines what they can consider, tools determine what they can reach, the policy and authority gate limits how the result may be used, and the decision record preserves the observable chain. The model is portable; its operational role is not carried inside the weights.

FIGURE 1B

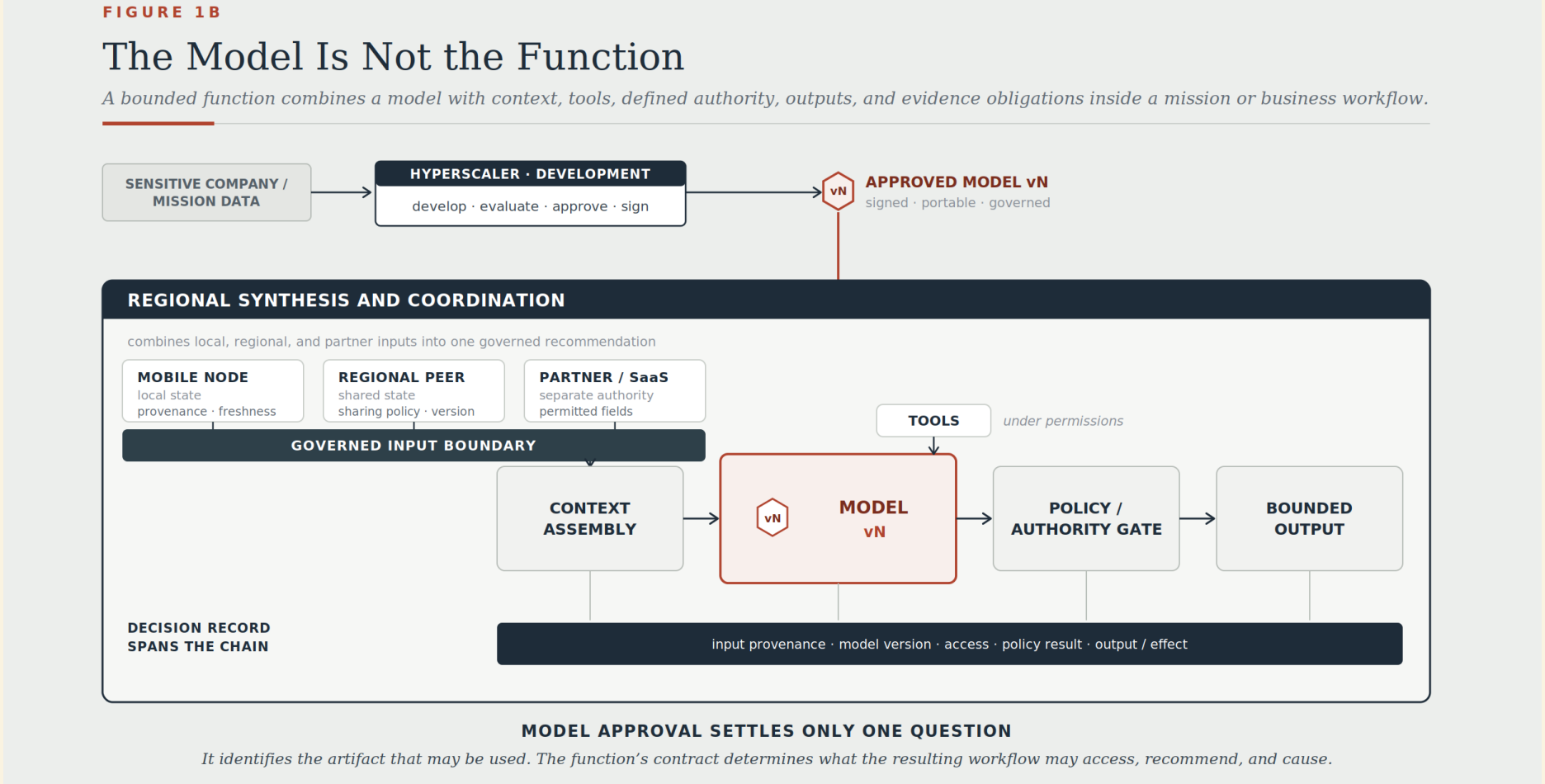

Figure 1B. *A model becomes part of a bounded operational function only through the context, tools, authority, output, and evidence obligations around it.*

Together the figures separate placement from implementation. Figure 1A holds the function constant and changes the environment. Figure 1B holds the model constant and shows the surrounding function. The execution contract governs the first relationship; the required contract defines the second.

The available environments form a trade space, not a ranking. Each offers a different combination of model capability, context, tools, connectivity, latency, capacity, data handling, authority, and record production. A placement is valid because that combination satisfies the required contract, not because the environment occupies a preferred tier.

The clauses describe different kinds of conditions: how well the function must answer, what it needs to answer, what its output may cause, and what it must record. A pairing can fail on any one of them. The five groups that follow are decision quality, context, permitted effects, degraded mode, and capacity and cost.

Each clause also specifies the record the pairing must produce. Evidence is not a sixth placement dimension; it makes compliance with the other five reviewable after the fact.

The decision-quality clause

The decision-quality clause states how well the function must perform for its judgment to remain usable. Placement can change the model, serving configuration, numerical precision, retrieval behavior, and operating conditions that produce the output. Returning the expected schema on time is not enough.

The required measures depend on the function. A screening function may need minimum recall for specified risk categories, calibration within a defined tolerance, robustness across expected data conditions, and abstention when required context or confidence is missing. The capability declaration must show that the proposed model, configuration, hardware, and serving software can meet those requirements together in the environment where they will operate. Different implementations can perform the same bounded function only while they remain inside the same required quality envelope.

That condition matters because a model does not carry the same capability into every environment. Compress it, give it a smaller context window or more limited retrieval, run it with different serving software, or keep it working on hardware that slows as it heats up, and its behavior may change. The function can still have the same name and return the same fields even though the quality of its judgment has shifted.

That is why evaluation has to reflect the pairing as it will actually run: the model, configuration, data, hardware, and sustained operating conditions together. Results from another setup are still useful, but they describe the model family. They do not show that this particular implementation can meet this particular contract.

Suppose Secondary Inspection Referral remains at a regional facility, but its serving implementation is replaced by a smaller quantized model to increase capacity and reduce cost. It responds faster and uses fewer resources. Those are real gains. But imagine that it now misses too many containers in a risk category the contract says it must reliably detect. Or its probability estimates no longer line up well with the outcomes they are meant to predict. In either case, the pairing is no longer valid, even though it has not moved and still returns the same fields.

The record needs to make that change visible. It identifies the model, its configuration, and the conditions under which it ran. It links to the relevant evaluation and calibration results, captures the uncertainty attached to the estimate, and shows whether Secondary Inspection Referral declined to answer when required. This record does not certify that the judgment was good. It gives a later assessment the evidence needed to determine whether the pairing remained within its required quality limits.

The context clause

A function can reason only over the data and tools its environment makes reachable. Reachable context depends on connected sources, their freshness and completeness, the network paths available at execution time, and the policies governing which records may be processed there. The required contract states

the minimum context and the limits on access. The capability declaration states what the environment can supply and under which conditions. Too little context can produce a confident judgment from an incomplete basis; excess access can expose records the function has no legitimate reason to use.

The hypothetical customs case makes that difference concrete through two possible placements.

At a national targeting center, Secondary Inspection Referral might draw on advance manifest data, importer history, watchlists, prior inspections, and trade-lane patterns assembled before the vessel reaches port. What it would not have is direct access to the physical container or to what inspectors observe. At the inspection lane, it might receive the X-ray or gamma image, radiation-portal readings, and the inspector's immediate observations, while having less access to the broader historical record.

Neither placement gives Secondary Inspection Referral a complete view. Each offers a different window onto the same container. A move between them could leave the output fields, timing, and interface unchanged while narrowing or otherwise changing the basis of the estimate.

Context includes more than whether a source is connected. Its age, completeness, jurisdiction, and relationship to the current case all affect what the function can draw from it. A synchronized watchlist may be reachable but stale. An importer record may omit activity held elsewhere. A tool may be connected but unavailable during an outage. Some records may not be copied or processed outside a jurisdiction, security boundary, or controlled environment at all.

When those limits prevent every source from gathering around one function, the design has to choose where the separate views may come together. Functions can run inside their respective boundaries and pass limited results to a permitted point of synthesis. A governed environment may receive selected evidence from both. Or the final comparison may remain with an authorized person who can review the separate records. Each choice creates its own functions, handoffs, and evidence obligations.

The record helps make that change visible. For each estimate, it shows which required context was available, what the function actually used, how current each source was, and what was missing. A Decision BOM can carry those facts forward. A later assessment compares them with the required contract to determine whether the basis was sufficient; the record itself does not make that decision.

The permitted-effects clause

The context clause asks what an environment can supply. The permitted-effects clause asks what surrounding systems may let the output cause. Each placement creates different low-friction integrations: a central function sits near systems of record and enterprise workflows; an edge function sits near gates, inspection queues, and equipment controls. Relocation can turn a deliberate cross-system connection into a local, inexpensive one.

Before relocation, reaching an operational mechanism may require a reviewed integration across network, system, or organizational boundaries. Afterward, the same connection can look like ordinary local composition: fewer hops, less orchestration, and less cost. That is why the required contract has to move with the function. A new placement may make additional actions easier to connect, but it does not expand what the output is authorized to cause.

Consider two hypothetical customs functions operating at different stages and on different containers. Before a vessel departs, one function might flag a particular container and recommend that it not be loaded. That container would not arrive at the domestic inspection lane on that voyage. Secondary Inspection Referral would assess containers that did arrive and might send one of them to a secondary-inspection queue. The decision to place a hold on that container, preventing it from moving onward, would remain with an authorized officer, as would any decision to open or seize it. The two functions could use the same version of the same model, with the same weights, yet their recommendations would be permitted to cause different things.

Suppose Secondary Inspection Referral is validly placed at the inspection lane. Its score sits close to the system officers use to place holds, and a new

integration connects high scores directly to that system. The change might look like an efficient way to remove a handoff. The model, placement, and score have not changed, but the system now turns a referral into an automatic hold. That is more authority than Secondary Inspection Referral's contract allowed.

For each output, the record should preserve the effect that followed, the authority under which it occurred, and the mechanism connecting the output to that effect. Without those fields, the automatic hold leaves no trace that the function's role changed.

This distinction also prevents a common ambiguity in inventories. Listing the model and its version cannot tell a reviewer what role its output played: whether it estimated risk, recommended review, referred a container to a queue, or authorized an action. Those are use-case verbs, not a universal taxonomy. Beneath them is the broader distinction developed in *Evidence Debt*: what did the function conclude, what was it allowed to see, and what did it ultimately cause? The decision-quality and context clauses govern the conclusion and its basis; the permitted-effects clause governs what that conclusion may cause. Two functions using identical model weights are still different bounded functions when their contracts permit their outputs to cause different things. The model is only one implementation component.

The degraded-mode clause

Every environment fails differently. A centrally placed function may reach larger compute pools, fresher shared data, and managed services while depending on the network. A locally placed function can continue through a network outage but may rely on synchronized records, local policy, and tighter resources. The placement decision therefore includes the failures the function must survive.

Now consider Secondary Inspection Referral at a hypothetical remote land crossing or smaller port where connectivity to national systems is thin or intermittent. Public-sector and defense practitioners often group such operating conditions under DDIL: denied, degraded, intermittent, or limited-bandwidth communications.

In a centralized design, the site sends the information needed for the assessment to a national service. That information might include the shipment manifest, importer history, relevant watchlist and policy results, and observations collected at the crossing. The national service estimates whether the container should be referred for additional inspection and returns that result before the local workflow must decide what happens next.

A local design places an approved model, the applicable rules, and authorized local or synchronized records at the site. It can continue producing referral estimates when the national connection is unavailable, then synchronize its decision records when connectivity returns. That continuity has limits: cached records grow older, some national sources become unreachable, and the local implementation may eventually have to narrow its output, abstain, or stop.

Both designs may satisfy the contract while the connection is healthy. During an outage, the centralized design can no longer deliver estimates to the site, while the local design can continue only within its declared limits. The PACE plan defines how Secondary Inspection Referral steps through those operating modes as conditions deteriorate.

The required contract must define that degraded region in advance:

- Which model, records, and policy versions may remain in use?
- What may Secondary Inspection Referral still return or cause?
- Which Decision BOM elements must it continue to record?
- When does its context become too stale to answer?

Degradation usually narrows what Secondary Inspection Referral may rely on, return, or cause. A referral estimate might still inform an officer during an outage while losing its ability to send a container directly to the secondary-inspection queue. Each estimate remains identifiable, tagged with the model, records, policy versions, and fallback conditions in force. Past a defined freshness limit, it may have to stop answering.

The fallback also needs an order. One familiar public-sector planning pattern is PACE: Primary, Alternate, Contingency, and Emergency. It is most often used for communications continuity, but the underlying discipline carries over: choose distinct fallback modes in advance, define what triggers each one, and do not treat a backup that depends on the same failed service as a genuine alternative.

Figure 2 applies that pattern to Secondary Inspection Referral. It is an illustrative design, not a description of current customs practice. The comparison adds what a communications plan alone would leave unspecified: the context available in each mode, what the function may return, what its output may cause, what must be recorded, and when that mode is no longer valid.

FIGURE 2

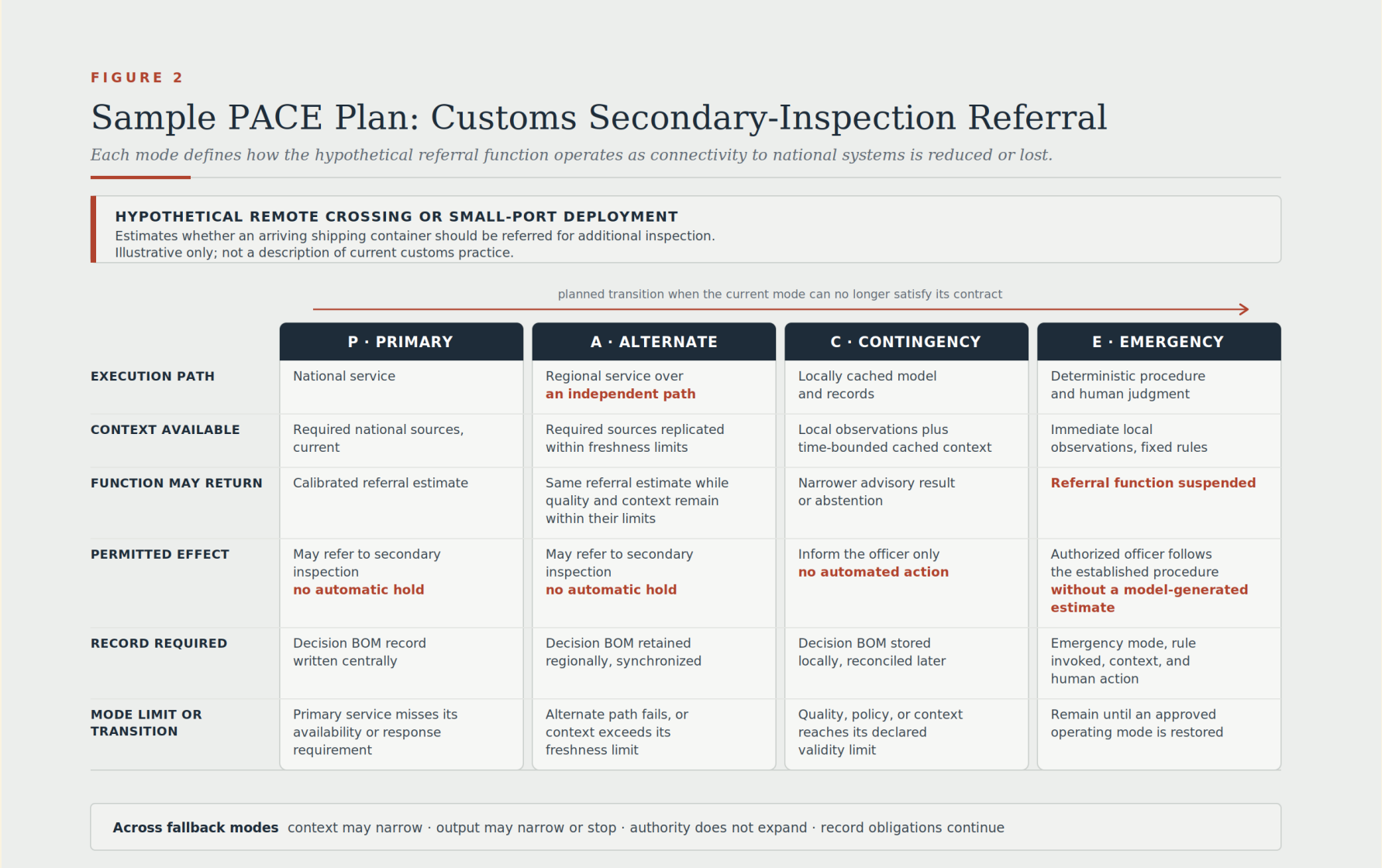

FIGURE 2

Sample PACE Plan: Customs Secondary-Inspection Referral

Each mode defines how the hypothetical referral function operates as connectivity to national systems is reduced or lost.

HYPOTHETICAL REMOTE CROSSING OR SMALL-PORT DEPLOYMENT
Estimates whether an arriving shipping container should be referred for additional inspection.
Illustrative only; not a description of current customs practice.

planned transition when the current mode can no longer satisfy its contract →

	P · PRIMARY	A · ALTERNATE	C · CONTINGENCY	E · EMERGENCY
EXECUTION PATH	National service	Regional service over **an independent path**	Locally cached model and records	Deterministic procedure and human judgment
CONTEXT AVAILABLE	Required national sources, current	Required sources replicated within freshness limits	Local observations plus time-bounded cached context	Immediate local observations, fixed rules
FUNCTION MAY RETURN	Calibrated referral estimate	Same referral estimate while quality and context remain within their limits	Narrower advisory result or abstention	**Referral function suspended**
PERMITTED EFFECT	May refer to secondary inspection **no automatic hold**	May refer to secondary inspection **no automatic hold**	Inform the officer only **no automated action**	Authorized officer follows the established procedure **without a model-generated estimate**
RECORD REQUIRED	Decision BOM record written centrally	Decision BOM retained regionally, synchronized	Decision BOM stored locally, reconciled later	Emergency mode, rule invoked, context, and human action
MODE LIMIT OR TRANSITION	Primary service misses its availability or response requirement	Alternate path fails, or context exceeds its freshness limit	Quality, policy, or context reaches its declared validity limit	Remain until an approved operating mode is restored

Across fallback modes context may narrow · output may narrow or stop · authority does not expand · record obligations continue

Figure 2. *A sample PACE plan for Secondary Inspection Referral. Each operating mode defines the available context, allowed output, permitted effects, required record, and transition conditions. In this example, contingency narrows the function and emergency suspends it.*

The important change is not simply the route used to reach a service. Each mode creates a different operating condition for Secondary Inspection Referral. Locally cached context may remain acceptable for a defined interval, after which it must narrow its output, abstain, or stop. If Decision BOM data cannot be synchronized immediately, it must be stored locally so the later record still shows which model, policy, context, and fallback conditions governed the decision. Loss of connectivity may narrow or suspend Secondary Inspection Referral; it does not give its output additional authority.

A placement optimized only for normal operation can satisfy its throughput, compute, and freshness targets while the network is healthy and produce nothing when the connection fails. Reconciliation of locally made decisions after connectivity returns is a subject for a future part.

The capacity and cost clauses

Capacity asks whether Secondary Inspection Referral can assess containers at the rate they arrive, including bursts from several active inspection lanes, and still return each estimate before the local workflow must decide what happens next. It also asks whether the deployed system can generate and retain the logs, source lineage, model and policy versions, output, and uncertainty needed to assemble the required Decision BOM record while that work continues.

That is why a single-request benchmark tells only part of the story. Requests can queue. Model loading and pressure on available memory can slow later estimates, cached information can be displaced, and Decision BOM data must be written while new requests continue to arrive. Testing therefore has to represent sustained, concurrent work rather than an otherwise idle system.

Cost needs separate treatment because it does not always mean a token bill. On an owned device or site server running an open-source model, the hardware may be a sunk cost, but power, storage growth, maintenance, and contention with other work remain. In a metered environment, inference, retrieval, tool use, network transfer, and record retention can all add usage charges. Autoscaling may preserve response time by adding resources, but without a ceiling it can also turn a burst of traffic into uncontrolled spend. The cost clause therefore has to name the limit that matters for the placement: fixed capacity, recurring spend, per-assessment cost, aggregate spend over time, or some combination. Whatever form that limit takes, a placement that stays within it only by dropping required logging, source lineage, or Decision BOM data has not met the contract.

Any candidate pairing would have to preserve the required decision quality at the volume of requests the site expects and within its resource and spending limits. If requests arrived faster than estimates could be completed, even a modest delay would grow into a queue. Caching, reuse, or precomputation might help when requests share context, but less so when each request requires different source data and fresh inference.

Updates do not all move on the same clock. A new model may need evaluation and approval before it can be deployed. A policy change may take effect at a stated time, while a retrieval index or local record set may have a maximum permitted age. A pairing remains valid only if the environment can receive, verify, and activate each kind of change within the window its contract allows. When it cannot, Secondary Inspection Referral must enter a defined fallback.

That requirement is separate from whether the pairing has enough capacity or is affordable. A site could stay within its compute, storage, network, and spending limits while running a model, policy, or data snapshot that is no longer permitted. Centralized fleet updates can make consistency easier, but a single rollout cadence may not fit an urgent or location-specific change. Local update paths can provide that flexibility while creating more versions to track and reconcile. The execution contract therefore has to define how quickly each kind of change must take effect, what happens when it cannot, and which versions governed each output. It covers the operating lifecycle, not only the instant of inference.

A pairing does not stay valid simply because it passed its checks at deployment. A model or serving update may change the quality of its estimates, while a lost connection to a data source can narrow the context available or leave it out of date. Heavy load or a network problem may keep the function in fallback longer

than the contract allows. Even an apparently helpful local integration can connect the output to an action it was never meant to cause. Through all of this, the function can remain in the same place and continue returning the same familiar fields.

Figure 3 follows one pairing through those four forms of drift. The execution home, invocation point, and output schema remain visibly unchanged while the conditions that made the pairing valid fall away.

FIGURE 3

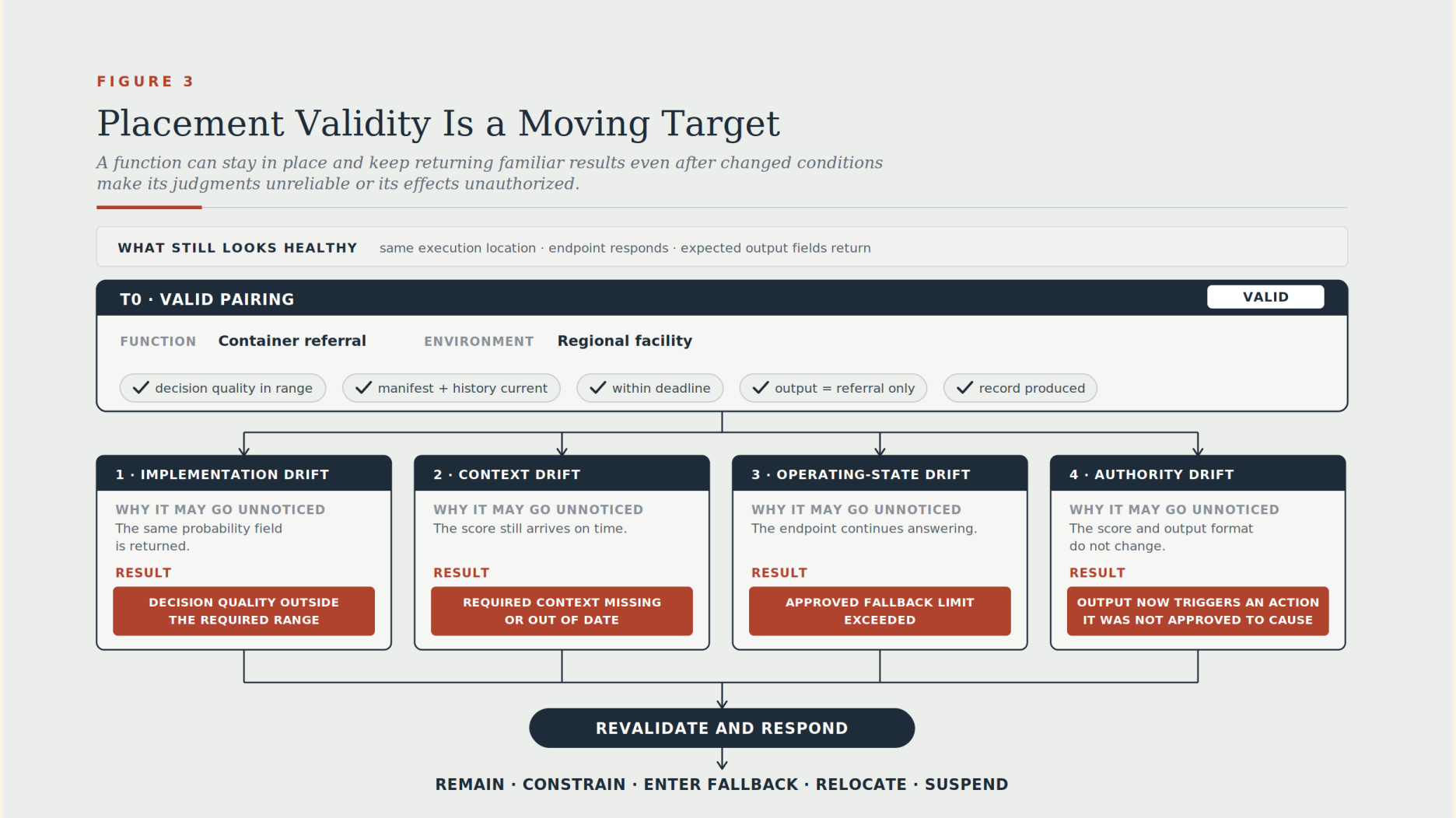

Figure 3. *Placement validity is temporal. Changes in implementation, context, operating state, or authority can require revalidation even when the function never leaves its environment.*

Passing the placement checks shows that the pairing satisfies the contract under the requirements and operating conditions considered at that time. If the requirements or operating conditions change, the pairing needs to be checked again.

That check may show that no action is needed. If something no longer lines up, the response should fit the problem: continue temporarily under the defined fallback, remove access or an integration the contract does not permit, move the function to another environment, or suspend it when no available placement is valid.

Maintaining validity requires a process that observes material changes, compares current conditions with the execution contract, and selects among those responses. That process may be operated by people through a change procedure or by software executing policy. Either way, the function that recommends or decides where other functions run becomes part of the architecture.

Placement Discipline

A function that recommends or decides where other functions run needs a required contract of its own. That contract states what information it needs about each environment, how current that information must be, and which relocations it may carry out, only recommend, or send for human approval. It also says what to do when telemetry, policy, or approval services are unavailable. One cautious response would be to leave current placements alone and pause new relocations until the missing information or service returns. Like the functions it manages, the placement function also needs enough capacity, resilience, and a reliable way to receive updates.

Two kinds of error matter. Moving a function unnecessarily can interrupt work, discard useful cached state, or increase cost. Failing to move when a pairing is no longer valid can leave the function using stale context, working without enough capacity, or waiting on a dependency it cannot reach. When information is uncertain or out of date, either mistake is possible. The placement function's contract should reflect their different consequences rather than treat every proposed move alike.

This does not require an endless chain of placement functions checking one another. Fixed rules and interlocks form a lower boundary. Mandatory inspection requirements, safety controls, and other prohibitions remain in force wherever a function runs; a probabilistic recommendation cannot weaken them.

At the upper boundary are decisions the institution has chosen to keep with accountable people. The placement function may recommend or carry out moves within its delegated scope, but changing where software runs does not enlarge that scope. Fixed rules below and reserved human authority above remain in force wherever the placement function itself runs.

So what does this add to systems that already place workloads well? It catches conditions ordinary placement controls may not see. A candidate environment can meet its latency, availability, and cost targets while running a model that falls outside its quality threshold, losing context that users still assume is present, or connecting an advisory score to an action the contract never permitted. Those are placement failures even when the workload itself appears healthy.

Nothing here requires replacing schedulers, load balancers, or placement policies. They can continue managing latency, capacity, availability, locality, cost, and infrastructure policy. What changes is the information available to that machinery, or to the process around it. Required decision quality and context, limits on access and effects, degraded behavior, and records owed become explicit placement constraints. The existing machinery can then do its familiar job against a fuller account of what successful execution means.

Start With One Function

Placement becomes much easier to govern once it is made inspectable. A required contract tells us what a valid judgment needs, and a capability declaration tells us what a candidate environment can actually support. Bringing them together gives a team an explicit account of why this function belongs

here, what must remain true, and what should happen when those conditions change.

Try it with one consequential function in a system you know. Keep its purpose fixed and compare the places it could run. Ask what each place lets the function know, which actions sit within reach of its output, how it behaves when a link fails or capacity tightens, and what record will be available later. Write down where the contract holds, where the function must narrow, where access or effects need to be blocked, and where no valid pairing exists. Those gaps give the team concrete choices: improve the environment, add a control, narrow the function, or stop it before its answer loses meaning.

Doing this once produces a placement decision. Operating AI requires doing it again as models, policies, context, and conditions change, and across many functions competing for the same environments. The next question is how to maintain that discipline: how to revalidate and relocate safely, account for decisions made under degraded contracts, and choose among functions when every contract cannot be satisfied at once. That is the path from a placement contract to an inference control plane.

Sources and Influences

This is a design essay rather than a literature review. The works below supplied concepts, evidence, or useful friction as the argument developed.

Earlier work in this series

- Stephen Alexander, *Operational AI Becomes a Distributed Systems Problem.* Establishes the decomposition of operational AI into bounded functions distributed across execution environments. cirrusly.me/Operational_AI_Distributed_Systems.pdf
- Stephen Alexander, *Operational AI in Search and Rescue.* Develops the relationship among function, placement, authority, and changing operational constraints. cirrusly.me/OperationalAI_SAR.pdf
- Stephen Alexander, *Evidence Debt.* Supplies the obligations concerning what a function concluded, what it was permitted to reach, what it caused, and what evidence remains afterward. cirrusly.me/Evidence_Debt.pdf
- Stephen Alexander, *From SBOM to Decision BOM: Why It Still Might Not Be Enough.* Distinguishes a complete decision record from an assessment of whether that record is sufficient. cirrusly.me/From_SBOM_to_Decision_BOM.pdf

Standards, frameworks, and research

- National Institute of Standards and Technology, *Artificial Intelligence Risk Management Framework (AI RMF 1.0).* Informs the treatment of deployment context, lifecycle risk, evaluation, documentation, and continuing revalidation. nist.gov/itl/ai-risk-management-framework
- European Parliament and Council of the European Union, *Regulation (EU) 2024/1689 (Artificial Intelligence Act).* A concrete example of regulation requiring technical documentation, event logging, record retention, risk management, and human oversight for covered AI systems. eur-lex.europa.eu/eli/reg/2024/1689
- World Wide Web Consortium, *PROV-Overview: An Overview of the PROV Family of Documents.* A general model of provenance as information about the entities, activities, and people involved in producing something that may later be assessed. w3.org/TR/prov-overview
- U.S. Department of Commerce, National Telecommunications and Information Administration, *The Minimum Elements for a Software Bill of Materials.* Supplies part of the bill-of-materials lineage behind the Decision BOM discussion. ntia.gov/files/ntia/publications/sbom_minimum_elements_report.pdf
- Hanwei Zhang, Andreas Schmidt, Sarah Sterz, and Holger Hermanns, *Constructing AI Accountability with Decision Bills of Materials.* Provides an external Decision BOM lineage and a concrete proposal for preserving the components and provenance of consequential AI decisions.
- Carlos Ibáñez Sánchez, *The Reasoned Ratification Report (IRR): Documented Human Oversight, Integrated Traceability and Control of AI-Assisted Judicial Decisions.* Offers an adjacent-domain proposal for making human review and adoption of an AI-assisted decision explicit and traceable rather than merely nominal.
- U.S. Department of the Army, *ATP 6-02.12: DoD Information Network–Army Planning Techniques.* Grounds the Primary, Alternate, Contingency, and Emergency planning pattern used in the degraded-mode example.
- U.S. Department of Defense Chief Information Officer, *Department of Defense Outside the Continental United States Cloud Strategy.* Grounds the DDIL operating context and the need to place data and computing closer to the point of need when reach-back connectivity is constrained.
- Jacob Wentz, Center for Strategic and International Studies, “Harnessing Edge AI to Strengthen National Security.” Public-sector context for local inference under disrupted or limited connectivity and the tradeoffs between edge and central execution.
- Kyle Miller and Andrew Lohn, Center for Security and Emerging Technology, *Onboard AI: Constraints and Limitations.* Grounds the discussion of how compute, memory, power, environmental conditions, and platform limits can change which models and safeguards are practical in a local deployment.
- Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q. Weinberger, “On Calibration of Modern Neural Networks.” Grounds the discussion of calibration as a property that must be evaluated rather than inferred from an output format or model label.
- Jemin Lee, Sihyeong Park, Jinse Kwon, Jihun Oh, and Yongin Kwon, “Exploring the Trade-Offs: Quantization Methods, Task Difficulty, and Model Size in LLMs From Edge to Giant.” Supports the claim that quantization can affect capabilities unevenly across tasks and therefore must be evaluated in the proposed configuration.

- Yinmin Zhong et al., “DistServe: Disaggregating Prefill and Decoding for Goodput-optimized Large Language Model Serving.” Demonstrates why serving capacity depends on workload, interference, resource allocation, and latency objectives rather than on an unloaded single-request benchmark alone.
- Hao Li et al., “LLMRouterBench: A Massive Benchmark and Unified Framework for LLM Routing.” Represents the quality-and-cost model-routing literature that this essay treats as one part of the larger function-placement problem.
- Cloudflare, *AI Gateway Features: Dynamic Routing and Spend Limits.* A current vendor example of gateway machinery for routing, fallback, rate control, and spending limits. Included as an implementation example, not as independent evidence of operational outcomes.