The Economics of the Agentic Factory: Capacity, Queuing, and Portfolio Optimisation

Executive summary

The operational scaling of an agentic factory introduces economic complexities that mirror traditional manufacturing and distributed computing, yet possess unique stochastic properties due to the probabilistic nature of autonomous agents. A small but growing organisation operating concurrent projects cannot merely maximise the number of active agents, as doing so typically leads to severe downstream congestion. The primary risk facing this architecture is not a deficit of agent productivity, but rather an overproduction of unverified, unintegrated activity that completely overwhelms human technical review, business approval, and systemic integration constraints. This comprehensive report establishes an advanced economic, capacity-planning, and portfolio-optimisation framework designed to maximise the delivery of accepted business outcomes rather than raw execution volume. By rigorously applying the Theory of Constraints, advanced queueing network principles, and modern FinOps practices, the analysis demonstrates that achieving true flow efficiency requires strict work-in-progress limits, expected-cost model routing, deterministic human-attention budgets, and dynamic capacity reserves.

Recommended direction

The organisation must fundamentally shift its operational paradigm from measuring raw computational activity—such as model calls executed, tokens generated, or speculative pull requests opened—to measuring the fully loaded cost and velocity of accepted business outcomes. Management must rigorously identify the system's active bottleneck, which frequently manifests not in the execution plane's compute capacity, but in human technical review, integration capacity, or manual business approval. Expanding upstream agent capacity without elevating these downstream constraints merely inflates queues, degrades end-to-end flow efficiency, and exacerbates costly rework loops. The recommended direction is to implement a pull-based economic admission controller governed by strict, mathematically derived work-in-progress limits. Furthermore, the factory must route tasks via expected-total-cost algorithms rather than relying on simple unit prices, maintaining protected capacity reserves to gracefully handle production incidents and stochastic variability.

Scope and assumptions

The operational environment described assumes a small but growing organisation orchestrating an agentic factory to service multiple simultaneous workflows. These workloads

span diverse domains including software engineering, research, and content production across various applications and repositories (e.g., project-alpha, project-beta). The factory orchestrates these tasks via specialised and generalist agent fleets (e.g., coding-fleet, research-fleet). Workflows rely on an execution plane for computation and a control plane for state management and orchestration. The underlying infrastructure abstracts multiple model providers and tool providers to maintain commercial leverage and routing flexibility. The economic analyses presented herein assume that agent output is fundamentally probabilistic; thus, failure and subsequent rework are inherent systemic features, not operational anomalies. All financial examples utilise generic pounds sterling (£) and assume a multi-provider strategy utilising distinct capability tiers, such as premium-model-class and economy-model-class.

Accepted-outcome economics

The foundational economic principle of the agentic factory is that the primary unit of value is the accepted outcome, defined exclusively as a completed result that satisfies explicit acceptance criteria, passes required verification, receives formal review, is successfully integrated, and delivers usable value without requiring immediate corrective rework. Treating intermediate outputs—such as tokens generated, tasks started, or draft outputs produced—as units of value creates a perverse incentive structure that obscures the true fully loaded cost of delivery.

The taxonomy of accepted outcomes varies by workflow class but retains a universally stringent requirement for definitive validation before economic value is recorded. A clear distinction must be maintained across the lifecycle: attempted output (the initial generation), technically completed output (code compiles or text is formatted), reviewed output (passed human or specialist-agent scrutiny), accepted output (formally approved for integration), deployed output, and value-realised outcome.

Cost and value taxonomy

A robust FinOps architecture requires capturing the comprehensive cost of an accepted outcome, extending far beyond the superficial metric of model inference pricing. The full cost taxonomy encompasses five primary categories.

Variable execution costs encompass input and output tokens, reasoning tokens where separately charged, cached input retrievals, external tool API calls, visual or audio generation, browser sessions, sandbox execution minutes, GPU minutes, storage operations, network transfer, and CI pipeline minutes.

Human costs, frequently omitted from naive economic models, are the most expensive component. These include operator review, business approval, clarification dialogues, technical and security reviews, integration, incident handling, postmortems, and policy maintenance.

Failure costs encompass the compounding expenses of failed runs, invalid outputs, retries, duplicated work, rework, rollbacks, delayed releases, abandoned branches, and unused generated artefacts.

Fixed platform costs consist of orchestration services, databases, observability platforms, backups, security tooling, provider minimum commitments, reserved capacity, and support

contracts.
Opportunity costs represent the invisible but critical losses incurred when high-value tasks are blocked by queues, when work awaits integration, when premium model capacity is starved, or when excessive work-in-progress delays market opportunities.
Simultaneously, raw outcome counts incorrectly treat trivial and high-value tasks equally. Value-adjusted outcome economics must incorporate dimensions such as revenue impact, cost reduction, time saved, risk reduction, strategic option value, learning value, and dependency-unblocking value. To avoid false precision, the factory should utilise ordinal value bands or expected value ranges rather than arbitrary, hyper-precise monetary assignments.

Cost attribution

Cost attribution dictates how financial consumption is assigned across the factory ecosystem—spanning the organisation, project, fleet, workflow, task, run, model provider, and accepted outcome.
Direct attribution efficiently maps variable execution costs to a specific task run. However, the factory also possesses substantial shared costs, such as common orchestration infrastructure, shared semantic memory, reusable research summaries, platform development, provider commitments, reviewer retainers, and incident reserves.
To prevent arbitrary overhead allocation from obscuring actual marginal economics, a dual-reporting structure is strictly required. Direct cost reporting accounts solely for the variable cost of the run, informing tactical capability-routing decisions. Fully loaded cost reporting includes a proportional or activity-based allocation of shared infrastructure and human operator retainers, providing a true measure of project-level profitability and portfolio health.

Theory of Constraints

Applying the Theory of Constraints to an agentic factory is paramount, particularly because upstream autonomous agents can generate output exponentially faster than downstream processes can consume it. The operational cadence must follow the five focusing steps.
First, the organisation must identify the constraint. In an agentic system, this is rarely model execution speed; it is typically operator approvals, technical review, integration capacity, testing, or release pipelines. Second, the constraint must be exploited. For example, if technical review is the bottleneck, the reviewer's time must be protected—they should only evaluate high-confidence, pre-validated outputs. Third, all other processes must be subordinated to the constraint. Upstream agents must be artificially paced to match the reviewer's capacity, explicitly preventing the buildup of massive review queues. Fourth, the constraint must be elevated, perhaps by adding more reviewer capacity, deploying specialist-agent reviewers, or automating portions of the testing phase. Fifth, when the constraint inevitably moves (e.g., from review to the CI pipeline), the cycle must repeat. Increasing upstream agent output without elevating the downstream constraint mathematically guarantees a reduction in total system performance.

Queueing-theory foundations

The agentic factory operates fundamentally as a stochastic queueing system. Service times are highly variable due to the probabilistic nature of generation, fluctuating provider API latencies, probabilistic retries, and the inherent unpredictability of human review times.

Little's Law, defined as $L = \lambda W$ (where L is average work in progress, λ is throughput, and W is average flow time), dictates that admitting more tasks into the system without increasing the completion rate will invariably increase the lead time for all tasks.[1] This principle demands that throughput be measured at the point of accepted-outcome completion, not at task initiation.

The Kingman formula (often referred to as the VUT equation) provides the mathematical proof for why running any factory component near maximum utilisation causes queue explosion.[2] For a generic G/G/1 queue, the expected waiting time E is approximated by:

$E \approx \left(\frac{\rho}{1-\rho}\right)\left(\frac{c_a^2+c_s^2}{2}\right)\tau$ Where ρ is utilisation, c_a and c_s are the coefficients of variation for arrivals and service times, and τ is mean service time. Because agentic workflows and human reviews possess inherently high variance (c_s^2 is large), pushing utilisation ρ towards 1.0 causes the waiting time to approach infinity.[1]

Furthermore, the factory operates as an open Jackson network with feedback.[3] In such a network, jobs that fail a quality gate return to an earlier node for rework.[5] The effective arrival rate at any node includes both external arrivals and internally routed rework. If first-pass acceptance rates drop, the internal rework traffic spikes, pushing nodal utilisation beyond 1.0 and causing the entire network to mathematically destabilise.[5]

End-to-end queueing network

To optimise flow, the factory must be modelled as a continuous network of interconnected stages. Work enters the backlog and proceeds to a readiness queue. Upon economic admission, it moves into active execution, which frequently involves fork-join synchronisation as main tasks split into subagents.

Outputs flow into automated verification. Rejections here form a rapid rework loop back to execution. Passing outputs enter specialist review or human approval queues. If rejected by a human, the work undergoes a more expensive rework loop or is classified for abandonment. Approved tasks enter the integration queue, followed by the release pipeline, culminating in formal acceptance and eventual value realisation. Analysing this full end-to-end queueing network reveals that optimising the execution phase in isolation is an economic fallacy.

Work-in-progress limits

Work-in-progress (WIP) limits are the primary systemic defence against queue explosion and factory degradation. Drawing from Kanban, manufacturing, and software delivery principles, WIP limits must control admitted work directly, not merely serve to display overloaded queues on a dashboard.

Limits should be applied dynamically across multiple dimensions: at the global organisation level to prevent aggregate budget overrun, at the project level to ensure fair capacity allocation, at the workflow stage (e.g., maximum unreviewed pull requests), and critically, at the human reviewer level to prevent cognitive overload. When a WIP limit is reached, the default platform behaviour must not be to create another hidden queue inside an agent's internal memory. Instead, the system must deterministically halt new task admission, block child-task creation, and ask autonomous agents to help clear the bottleneck or pause entirely.

Task admission and prioritisation

Every transition from the backlog into active WIP must represent an explicit economic decision. The economic admission controller must evaluate expected value, readiness, explicit acceptance criteria, dependency status, expected cost, required model class, and current incident reserve impact before starting a task.

Prioritisation is governed by a value-adjusted Cost of Delay (CoD) methodology, specifically adapting Weighted Shortest Job First (WSJF) for stochastic agent tasks. The formula used is:

$$\text{WSJF} = \frac{\text{Expected Value} \times \text{Acceptance Probability}}{\text{Estimated Lead Time} + \text{Rework Penalty}}$$

This approach prevents large, highly uncertain tasks from permanently starving rapid, high-value, deterministic work. The priority record must be explicitly machine-readable, exposing the estimated cost of delay, urgency, risk reduction, and expected human attention, explicitly avoiding one opaque, unauditable priority score.

Human-attention capacity

Human attention is the most strictly constrained service resource in the agentic factory. Human work includes business decisions, technical and security reviews, incident response, and policy maintenance. Human attention is explicitly not equivalent to clock time; it is highly susceptible to interruptions, context-switching penalties, cognitive complexity, and fatigue.

The factory must employ a formal Attention-Demand model:

$$\text{Attention Demand} = (\text{Base Review Time} \times \text{Complexity Factor} \times \text{Risk Factor}) + \text{Context-Switch Cost}$$

Capacity planning for the primary operator, technical reviewer, or business approver must reserve substantial time for unexpected escalations, strategic decisions, and system maintenance. Scheduling all theoretical human working hours to handle routine approvals mathematically guarantees that the bottleneck will starve upstream flow and cause systemic failure.

Approval and review bottlenecks

A rapidly rising human approval queue frequently indicates a poorly designed autonomy

architecture, inadequate task framing, or over-sensitive risk thresholds, rather than a lack of operator effort. The platform must define an explicit approval-demand budget, such as a maximum number of routine approvals per day or a strict limit on after-hours interruptions. When the approval budget is exceeded, the system must automatically trigger a structural investigation into policy design and duplicate requests. While batching approvals can reduce context-switch costs and notification burdens, it risks lowering the attention per item, hiding severe outliers, and delaying urgent work.[6] Optimal batch size must be strictly modelled based on risk homogeneity and review complexity.

Integration and release bottlenecks

Increasing parallel coding capacity through agent scaling actively harms delivery throughput if integration and release capacity remain fixed. Integration requires resolving semantic code conflicts, merging database migrations, running system-wide tests, and coordinating releases. Metrics such as integration queue length, semantic conflict rate, mean integration time, and the age of integration-ready branches serve as critical early warning signals. When release windows, canary durations, or manual production access approvals constrain the system, adding more upstream execution agents simply inflates the inventory of undeployed code, thereby increasing the probability of complex, cascading merge conflicts and severe rework.

Parallelism and fork-join economics

Agentic workflows frequently employ speculative or decomposed parallelism, splitting work among multiple subagents and subsequently synchronising the results.[7] This creates a complex fork-join queueing dynamic.[8]

The primary economic risk of a fork-join architecture is the severe synchronisation penalty: a job is only completed when its *slowest* branch finishes.[8] The expected maximum delay of K parallel branches grows logarithmically, scaled by the harmonic number H_K (where $H_K = \sum_{i=1}^{K} \frac{1}{i}$).[8] Applying the Nelson-Tantawi approximation, the response time T_K of the fork-join system is substantially longer than the average of a single branch, and this penalty increases as load increases.[8]

The net economic value of parallelism is calculated as:

$$\text{Net Parallelism Value} = \text{Lead-Time Reduction Value} - (\text{Coordination Cost} + \text{Conflict Cost} + \text{Duplicate Work Cost} + \text{Integration Cost})$$

Parallelism is economically viable when tasks are deeply independent, isolated by strict modular contracts, and independently testable. It is highly destructive when agents must modify shared architecture or require a final, unified creative judgement.

Speculative execution

Speculative execution—such as dispatching two distinct models to attempt the same difficult reasoning task simultaneously, or having a fast economy model begin work while a premium model evaluates the plan—can lower tail latency and improve success probability. However, it

incurs severe duplicated compute, review, and selection costs, alongside the risk of inconsistent evidence. Speculative execution is economically justified only when the cost of delay (e.g., tail latency during a critical production incident) massively outweighs the marginal inference cost of the duplicate runs, and when the task possesses deterministic, automated verification logic.

Independent-review economics

Independent review by a separate model, another agent role, or a deterministic tool generates value by intercepting errors before they reach human operators or production systems.

$$\text{Expected Review Value} = (P(\text{Detecting Material Error}) \times \text{Expected Avoided Loss}) - (\text{Review Cost} + \text{Delay Cost})$$

Crucially, two reviews are not mathematically independent merely because two agent instances performed them. Correlation of errors occurs rapidly if the agents use the same underlying foundation model provider, identical prompts, or the same retrieved context. True independent review requires methodological, prompt, or provider diversity, and should be scaled deeply based on expected loss, irreversibility, and production impact.

Premium-versus-economy model allocation

To optimise the provider portfolio, tasks must request abstract capabilities and economic boundaries rather than hard-coded model names.[10] A provider-neutral model-class system must delineate between economy, standard, premium, specialist, low-latency, and local/private tiers.

Cost-aware routing must classify the request, apply workload-specific quality constraints, and select the lowest-cost model expected to satisfy them safely.[10] The platform requires an expected-cost routing model:

$$\text{Expected Total Cost} = \text{Initial Run Cost} + (P(\text{Failure}) \times \text{Expected Recovery Cost}) + (P(\text{Rework}) \times \text{Expected Rework Cost}) + \text{Review Cost} + \text{Delay Cost}$$

An economy-model-class may possess a drastically lower initial unit price but a vastly higher expected total cost if its high failure rate requires multiple retries, intensive human review, and complex downstream integration rework.[10] Conversely, deploying a premium-model-class for deterministic extraction is pure waste. A safe implementation relies on explicit quality floors, pinned models, and cheap deterministic output validation.[10]

Provider portfolio optimisation

Providers must be treated as a dynamic portfolio rather than isolated vendors. Routing solely to the cheapest current provider invites severe provider arbitrage risks, including unstable rate limits, sudden quality degradation, hidden retry costs, and outage correlation.[6] Concentration risk must be aggressively monitored via dashboards tracking the percentage of critical workflow spend assigned to any single provider, ensuring that fallback routing logic and adapter maturity remain continuously tested and viable.

Provider commitments and concentration

Purchasing provider commitments—discounts offered in exchange for guaranteed volume—requires rigorous scenario analysis.[11] A spend-based commitment guarantees a discount in exchange for a monetary minimum, allowing flexibility across a provider's portfolio, whereas a usage-based commitment requires consuming specific resource units (e.g., specific compute instances).[12]
The break-even analysis for a commitment discount must evaluate:

$$\text{Effective Commitment Value} = \text{Expected Discount Savings} - (\text{Unused Commitment Risk} + \text{Lock-In Cost} + \text{Reduced Routing Flexibility})$$

Commitment underutilisation directly penalises the factory as unused amounts are billed.[11] The organisation should target right-sized coverage, heavily resisting the urge to commit to 100% of baseline demand, as locking in volume severely degrades multi-provider routing flexibility during periods of rapid model capability shifts.[12]

Retry, checkpoint, abandonment, and rework economics

Blind, infinite retries are economically ruinous. A retry should only be executed if the system explicitly alters the underlying strategy (e.g., prompt adjustment, tool adjustment, or higher-capability model escalation) and if the expected value calculation remains positive:

$$\text{Expected Value of Retry} > \text{Expected Retry Cost} + \text{Delay Cost} + \text{Risk Cost}$$

Because successive similar failures suffer from rapidly diminishing marginal returns, strict retry ceilings must be enforced based on task classification, historical recovery rates, and remaining budget limits.
Furthermore, explicit abandonment rules must be established. Work must be aggressively discarded when its expected value falls beneath its expected completion cost, or when dependencies disappear, directly countering sunk-cost bias. Rework economics are tracked via the Rework Ratio (Cost of Rework divided by Total Delivery Cost). The First-Pass Acceptance Rate acts as the leading indicator of workflow health, evaluating whether templates and agents are truly capable or merely relying on brute-force trial and error.

Marginal value of additional agents

The fundamental assumption that adding agents proportionally increases throughput is false. The Marginal Agent Value must be calculated precisely as the change in Accepted Outcome Value divided by the change in Total Factory Cost. Clear signs of agent saturation include rising human review queues, falling first-pass acceptance rates, a surge in semantic merge conflicts, a higher abandoned-work rate, and an increasing volume of operator interruptions. When the system becomes saturated, the correct economic intervention is lowering the WIP limit and elevating the integration bottleneck, not expanding the agent fleet.

Fleet sizing and capacity planning

Fleet sizing cannot be calculated using average task arrivals alone due to the severe penalties of stochastic variability in service times. Sizing requires modelling demand rates against service

rates, target lead times, review capacity, incident reserves, and budget constraints. Planning horizons must separate real-time admission throttling from weekly routing adjustments, monthly commitment optimisations, and quarterly infrastructure scaling.[15] Demand forecasting must utilise quantile forecasts, backlog-based projections, or Monte Carlo simulations to represent genuine uncertainty, explicitly avoiding false point precision.[15]

Incident-capacity reservation

The factory must maintain a strictly protected capacity reserve for incidents, modelled heavily on Site Reliability Engineering (SRE) principles regarding headroom.[15] If 100% of the platform's resources are consumed by routine workloads, an urgent security incident, provider outage, or rapid task failure will displace work unpredictably, causing cascading queue failures.
The ideal configuration targets a "right-sized" baseline utilisation of 70% to 85%, leaving 15% to 30% of system capacity—encompassing human attention, premium-model-class tokens, and budget—strictly reserved for incidents and variance absorption.[15] An explicit incident pre-emption policy must dictate which routine tasks are paused gracefully, which queues lose capacity, and how state is safely retained during emergencies.[17]

Project portfolio allocation

When multiple projects compete for constrained capacity, dynamic value-based allocation is required rather than fixed equal project quotas. Projects are prioritised based on strategic alignment, revenue potential, dependency-unblocking value, and the probability of successful outcomes. Highly valued projects must still be governed by minimum service levels and explicit ageing priorities for other queues to prevent them from indefinitely starving low-priority but necessary operational maintenance work.

Flow, throughput, lead-time, and yield metrics

Economic visibility relies fundamentally on accurate flow metrics:

- **Flow Efficiency**: Active Work Time divided by Total Lead Time. Consistently low flow efficiency indicates severe queueing delays rather than slow execution agents.
- **Outcome Yield**: Accepted Outcomes divided by Tasks Admitted.
- **Cost per Accepted Outcome**: Total Attributable Factory Cost divided by Number of Accepted Outcomes.

The system must track queue-age metrics (focusing on the median and 90th percentile, rather than the average) to identify hidden blockers, unclear ownership, and neglected task queues. Lead-time metrics must strictly separate active working time from queue waiting time and customer-controlled delays.

Waste, context, and memory economics

Lean operations principles apply directly to the agentic factory. Waste manifests economically as unnecessary task creation, excessive context loading, duplicate research, oversized batches, and premature model overqualification.

Context-loading economics measure the fixed financial and latency cost of injecting large repository contexts, historical decisions, or semantic embeddings into every run. Excessive task fragmentation forces the system to repeatedly pay this context-loading tax. Conversely, maintaining a curated, shared organisational memory reduces rework, repeated research, and onboarding time, though it incurs persistent vector-storage and automated curation costs.

Automation and platform-investment economics

Investments in custom orchestration, provider abstractions, and dedicated evaluation suites must be rigorously evaluated against standard build-versus-buy economics.

$$\text{Automation Value} = \text{Expected Labour Savings} + \text{Error Reduction Value} + \text{Lead-Time Reduction Value} - (\text{Build Cost} + \text{Maintenance Cost})$$

The factory should heavily avoid automating unstable or poorly understood workflows prematurely. Buying managed model gateways and observability tools is often vastly cheaper over a two-year horizon than absorbing the continuous engineering opportunity cost of maintaining custom provider adapters internally.

Forecasting and simulation

Before executing major capacity changes or fleet expansions, the platform architect should employ Monte Carlo simulation to model distributions for task arrivals, task complexity, model success probabilities, service times, and incident arrivals. This outputs expected lead-time percentiles, queue lengths, and operator overload probabilities. While a lightweight operational digital-twin concept is theoretically powerful for simulating WIP limit changes and provider commitments, it must be completely deferred until basic deterministic data quality and comprehensive event instrumentation are firmly established.

Economic dashboards and alerts

The dashboard architecture must act as a precise decision-support command centre, strictly prioritising action over decorative charts.

- **Constraint Dashboard**: Identifies the current probable bottleneck, showing affected projects, buffer status, and recommended actions.
- **Queue Health Dashboard**: Displays arrival rates, service rates, 90th percentile queue age, and predicted threshold breaches.
- **Provider Economics Dashboard**: Tracks accepted-outcome costs, first-pass acceptance, retry rates, and concentration limits.

Economic alerts must be configured strategically. Over-alerting at 60% utilisation creates severe alarm fatigue, while under-alerting at 95% precludes smooth reactive capacity injections.[15] The critical threshold for action should generally reside in the 85% to 90% range.[15]

Policies and economic invariants

The platform must enforce unyielding, deterministic economic invariants to guarantee stability. No task may enter active WIP without formal readiness and budget evaluation. WIP limits must be enforced outside the agents at the platform level. Agents cannot unilaterally increase their

own budget ceilings or create unlimited child tasks. Retries require a definitively classified failure and a changed strategy. Premium capacity must remain available for defined critical work, and incident capacity cannot be consumed by routine workflows. Safety and required verification loops cannot be bypassed merely to temporarily inflate throughput.

Schemas and data model

To operationalise these economics, the platform relies on strict, machine-readable generic schemas.

Schema Type	Core Fields Required
Task Economic Profile	Expected value, value confidence, urgency, cost ceiling, expected duration, attention demand, risk, acceptance probability, required capability.
Model-Routing Decision	Task ID, selected model class, expected total cost, alternatives evaluated, assumptions, confidence.
Retry Decision	Prior attempt ID, failure classification, changed strategy, incremental budget, expected benefit.
Human-Attention Record	Reviewer ID, base review time, complexity factor, context-switch penalty, outcome.
Accepted Outcome	Task ID, acceptance event timestamp, validation authority, direct cost, fully loaded cost.

Conformance and adversarial tests

The factory must be rigorously subjected to adversarial testing to prevent systemic metric gaming. For instance, agents might attempt to bypass project WIP limits by splitting work into hundreds of micro-tasks. The system must test for "retry spirals" where an agent repeatedly consumes budget without materially altering its strategy. Audits must ensure that human review time is never omitted from the fully loaded cost ledger, that abandoned work is fully accounted for in portfolio overhead, and that cost forecasts cannot be manipulated through falsely optimistic success estimates.

Required comparison matrices

Queue Disciplines

Discipline	Throughput Impact	Lead Time Impact	Risk / Complexity
FIFO	Average	Average	Low (ignores economic value)
Shortest Job First	High for small tasks	Poor for large tasks	Medium (requires accurate estimation)
Highest Value First	High economic yield	High variance	High (starvation risk for low-value tasks)
WSJF (Cost of Delay)	Optimal	Optimal	High (requires complex stochastic value calculations)
Ageing Priority	Low	Reduces tail latency	Low (prevents queue starvation)

Model-Routing Strategies

Strategy	Cost Profile	Quality / Rework Profile	Ideal Use Case
Premium First	High	High first-pass acceptance	High-risk, irreversible business decisions
Economy First	Very Low initial	High rework and review risk	Deterministic, highly verified batch work
Economy +	Medium	Good	Large-scale content

Premium Review			generation with safety needs
Adaptive (Expected Cost)	Optimal	Optimal	Mixed, multi-project continuous operations

Parallelism Strategies

Strategy	**Lead Time**	**Cost / Waste**	**Rework / Conflict Risk**
Serial	Slowest	Lowest	Lowest
Task Decomposition	Fast	Medium	Medium (requires strict API/contracts)
Speculative Duplicates	Fastest	Highest (100% waste on losers)	Low
Dynamic Parallelism	Optimal	Medium	Medium (adjusts based on recent conflict rates)

Capacity Policies

Policy	**Utilisation**	**Queue Delay**	**Incident Resilience**
Maximum Utilisation	Approaching 100%	Exponentially High	None (brittle)
Fixed WIP	Controlled	Bounded	Medium
Dynamic WIP	Highly Controlled	Optimal	High
Incident Reserve (20%)	Baseline 80%	Low	Maximum

Provider Commitments

Type	Upfront Cost	Routing Flexibility	Unused Capacity Risk
On-Demand	None	Maximum	None
Volume Tier Discount	Low	High	Low
Spend-based Commitment	Medium	High (across vendor's portfolio)	Medium
Usage-based Commitment	High	Low (locked to specific models)	High

Review Strategies

Strategy	Speed	Cost	Error Detection Probability
Deterministic Check	Instant	Near Zero	Perfect for syntax; zero for semantic logic
Same-Provider Review	Fast	Low	Poor (high correlation of errors)
Cross-Provider Review	Fast	Medium	Good (methodological diversity)
Specialist-Agent Review	Fast	High (context load)	Very Good
Human Review	Very Slow	Extremely High	Excellent

Reference architecture

The system design for the agentic factory control plane relies on loosely coupled, event-driven components enforcing economic constraints at every boundary.
--> --> [Economic Admission Controller] | (Evaluates Budgets, Constraints & WIP Limits) | v
| [Incident Capacity Manager] -reserves-> <--> [Provider Portfolio Manager] | [Execution Plane / Agents] | (Automated Verification Gate) | <--> | [Integration Queue] |
|
| The architecture ensures that task flow, cost flow, and attention flow are explicitly measured. Provider-routing decisions and retry decisions are heavily mediated by the Capability Router, while feedback loops immediately inform the Admission Controller of downstream integration bottlenecks.

Build-versus-buy analysis

Capability	Recommendation	Rationale
Workflow / Orchestration Engine	Buy	Mature open-source and managed cloud options exist; building incurs immense maintenance debt and distraction.
Model Gateway / Abstraction	Buy	Vendor APIs, tokenisation methods, and features shift constantly. Managed gateways provide instant multi-provider access.
FinOps Cost Analytics	Buy	Cloud billing ingestion and normalisation is highly complex and fully solved by existing market tools.[14]
Task-Value Prioritisation	Build	Unique to the business context; off-the-shelf tools lack domain-specific outcome definitions and CoD weighting.

WIP & Queue Control Policy	Build	Core operational IP of the factory; requires deep, real-time integration with internal routing logic and human-attention models.
Capacity Simulator	Buy / Defer	Building a custom discrete-event simulator is prohibitively expensive for a small team.

Architecture decision records

ADR	Context & Decision	Expected Economic Effect
ADR-001: Primary Economic Unit	**Context:** Activity metrics hide waste. **Decision:** Define accepted outcome as the primary economic unit.	Prevents optimising for intermediate outputs and accurately exposes true rework costs.
ADR-002: WIP Limits	**Context:** High utilisation causes queue explosion.[2] **Decision:** Enforce hard WIP limits at project and stage levels.	Enforces Little's Law to prevent queue inflation and safeguard human attention.
ADR-003: Model Routing	**Context:** Fixed models cause lock-in and waste. **Decision:** Use expected-total-cost routing based on abstract capability classes.[10]	Optimises true delivery cost by factoring in the probability of failure and subsequent rework.
ADR-004: Human Attention	**Context:** Operators are burning out on approvals. **Decision:** Measure human attention as a constrained service resource; use	Acknowledges that business/technical approvals are the actual factory bottlenecks, preserving capacity for

	exception-driven approvals.	strategy.
ADR-005: Incident Reserve	**Context:** Incidents displace routine work unpredictably. **Decision:** Maintain a fixed 20% incident capacity reserve.[15]	Ensures systemic stability during provider outages or security escalations without starving production.
ADR-006: Retry Ceilings	**Context:** Agents loop indefinitely on failures. **Decision:** Retries require a classified failure and a changed strategy.	Prevents infinite loop waste and forces timely task abandonment.

Phased implementation roadmap

Phase A — Measurement foundations Objective: Establish economic visibility. Implement accepted-outcome definitions, task-level cost attribution, queue-stage timestamps, and direct model/tool cost capture. Project attribution and the basic outcome registry must be launched.*Safety guardrail:* Do not alter existing workloads or routing logic during this observation phase.
Phase B — WIP and queue control Objective: Stabilise system flow. Implement readiness states, economic admission control, project and stage WIP limits, queue priorities, and queue-age alerts.*Validation:* Success is measured by a direct, sustained reduction in the integration queue length and median lead time.
Phase C — Provider and routing economics Objective: Optimise variable execution cost. Implement capability-based model classes, provider cost ingestion, accepted-outcome provider metrics, expected-total-cost routing [10], stringent retry policies, and provider concentration monitoring.
Phase D — Capacity planning Objective: System resilience. Implement demand forecasts, formal human-attention capacity tracking, dynamic incident reserves [15], fleet sizing, and basic discrete-event simulation scenarios.
Phase E — Portfolio optimisation Objective: Maximise strategic value. Implement Cost of Delay (WSJF) task prioritisation, dynamic project capacity allocation, marginal-agent analysis, and formal provider commitment break-even analysis.[12]
Phase F — Adaptive optimisation Objective: Continuous improvement. Implement learned contextual routing, dynamic WIP adjustments, automated experiment design, and advanced portfolio optimisation.*Deferred capability:* Digital-twin simulation is only implemented here, once deterministic data is flawless.

Ordered implementation backlog

Outcome Definitions & Cost Collection

1. Define accepted outcome triggers for software, research, and content pipelines.
2. Implement telemetry to capture execution timestamps (from admission through to acceptance).
3. Build direct cost collector for token usage, tool API calls, and CI minutes.
4. Establish fully loaded cost model including proportional platform and human overhead.
5. Create central Outcome Registry database schema.

Queue Instrumentation & WIP Controls 6. Instrument backlog, review, and integration queues with length and 90th percentile age metrics. 7. Implement hard platform-level WIP limits per project. 8. Implement WIP limits per operational stage (e.g., maximum unreviewed PRs). 9. Build Economic Admission Controller to halt tasks when WIP is breached. 10. Develop queue-age alerting for tasks blocked beyond standard SLA limits.

Human Attention & Review 11. Define the Attention-Demand schema (base time, complexity factor, context-switch). 12. Build Human-Attention dashboard tracking daily operator utilisation versus scheduled capacity. 13. Implement exception-driven approval policies to reduce routine operator load. 14. Develop review batching logic for low-risk, homogenous tasks.[6]

Provider Routing & Retries 15. Map existing hard-coded model requests to generic capability classes. 16. Build capability router to select models based on expected total cost.[10] 17. Implement fallback routing for provider latency/outage degradation. 18. Establish retry ceiling policies based on task risk and budget limits. 19. Enforce "changed strategy" rule: block retries that do not alter prompt, tool, or model class.

Parallelism & Incident Reserve 20. Implement sub-agent creation limits to prevent fork-join queue explosions.[8] 21. Define incident pre-emption protocols for graceful task pausing.[17] 22. Configure a 20% dynamic capacity reserve (models, budget, CI) explicitly protected from routine workloads.[15] 23. Build metric tracking for semantic merge conflicts caused by parallel agents.

Project Prioritisation & Portfolio 24. Implement Cost of Delay (WSJF) formula in task metadata. 25. Develop Project Portfolio dashboard highlighting flow efficiency and outcome yield. 26. Establish provider concentration limits (e.g., maximum 60% critical spend on one vendor). 27. Build financial model for evaluating spend-based provider commitments.[11]

Forecasting, Simulation & Policy 28. Build Monte Carlo simulation for predicting end-of-month budget variance. 29. Create digital twin subset to simulate WIP limit changes on integration lead time. 30. Implement deterministic policy engine to reject tasks exceeding cost ceilings. 31. Establish automated adversarial testing for metric gaming and retry spirals. 32. Define ADR governance process for future routing algorithm updates.

Final recommendations

Recommended economic operating model The factory must optimise strictly for accepted business outcomes. Activity metrics (e.g., agent hours, lines of code, tokens generated) must be aggressively deprecated as performance indicators. Cost per accepted outcome, fully incorporating failure waste and human review time, is the singular financial truth.

Recommended constraint-management method The active bottleneck must be identified

via queue-age and integration backlog metrics, not by observing the busiest component. The constraint (most often human integration or technical review) must be protected from low-confidence upstream outputs and elevated through specialist automation. Upstream agent generation must be unequivocally subordinated to the constraint's throughput rate.

Recommended WIP policy Strict WIP limits must be enforced at the organisational, project, and workflow stage levels by an economic admission controller. When limits are reached, new task admission must stop entirely. Dashboards alone are insufficient; limits must be computationally binding to prevent queue explosion.

Recommended human-attention model Human attention is a finite, highly sensitive service resource. It must be explicitly budgeted using attention units that factor in cognitive complexity and context-switching penalties. Schedulable routine approvals must never exceed 70% of human capacity, reserving the remainder for strategic policy design and incident resolution.

Recommended task-priority model Admission and scheduling must be governed by a value-adjusted Cost of Delay (WSJF) framework. Expected value, acceptance probability, dependencies, and risk dictate priority, mathematically preventing high-effort, low-value work from starving critical strategic initiatives.

Recommended model-routing policy Models must be routed dynamically based on capability classes and the Expected Total Cost of the outcome, factoring in the statistical probability of failure and rework.[10] Economy-model-classes should handle deterministic, well-verified batch work, while premium-model-classes should be strictly reserved for high-risk, deeply contextual, or irreversible tasks.

Recommended provider-portfolio policy The platform must maintain independence through provider-agnostic capability routing. Strict concentration limits must be enforced to prevent lock-in. Provider commitments should heavily favor spend-based flexibility over usage-based rigidity, and must only be executed after rigorous break-even analysis.[12]

Recommended retry policy Retries must be strictly capped and evaluated economically. A retry is only permitted if the strategy formally changes (e.g., tool adjustment, model escalation) and the expected value of success still outweighs the compounded cost of delay. Sunk-cost abandonment must be enforced automatically.

Recommended parallelism policy Parallel execution is only authorised for highly isolated, independently verifiable modules with strict programmatic contracts. Because of the severe fork-join synchronisation penalty [8], speculative execution is limited to tasks where lead-time reduction explicitly outweighs the 100% waste generated by discarded branches.

Recommended independent-review policy Cross-provider or multi-method independent review is economically justified for high-risk outcomes where the cost of production failure is catastrophic. Correlated reviews (same prompt, same foundation model family) must be disallowed as they provide dangerous false confidence.

Recommended incident-reserve policy A minimum 20% capacity reserve—encompassing premium tokens, human attention, integration slots, and budget—must be cordoned off for incident response.[15] Formal pre-emption rules must dictate how routine workflows gracefully pause when emergencies consume this reserve.

Recommended cost-per-accepted-outcome model The fully loaded cost per accepted

outcome must divide the total attributable factory cost (including inference, CI, human review, and all failed/abandoned attempts) by the number of outcomes accepted at the final quality gate. Direct costs should be tracked separately to inform routing.

Recommended capacity-planning model Fleet sizing and reviewer capacity must be planned using stochastic demand forecasting, accounting for variance. Linear planning averages fail entirely in agentic systems; buffers must exist to absorb routine service-time spikes without breaching SLAs.

Recommended provider-commitment policy Commitments must only be purchased when the effective commitment value remains positive under rigorous sensitivity analysis, accounting for a potential 30% reduction in demand or a sudden requirement to shift to a competing model architecture.[12]

Minimum safe baseline Before multiple projects operate concurrently, the organisation must have implemented accepted-outcome definitions, task-level cost attribution, project WIP limits, exception-driven human approval queues, hard retry ceilings, and an incident capacity reserve. Without these, multi-project scaling is economically unsafe and mathematically unstable.

Controls that may be deferred Advanced capabilities such as automated contextual bandit routing, full digital-twin simulations, and dynamic algorithmic portfolio rebalancing can be safely deferred until the minimum safe baseline yields six months of clean, deterministic queueing data.

Top fifteen economic and capacity risks

Rank	Risk	Probability	Financial Impact	Throughput Impact	Detectability	Mitigation Urgency
1	Downstream integration bottleneck starvation	High	High	Critical	Low	Immediate
2	Unlimited retry spirals consuming budget	High	Critical	Medium	High	Immediate
3	Operator attention	High	Medium	Critical	Medium	Immediate

	exhaustion / alert fatigue					
4	Fork-join synchronisation penalty (parallel agents)	High	Medium	High	Low	Immediate
5	WIP explosion due to unchecked admission	High	High	High	High	Immediate
6	Premature commitment to obsolete provider	Medium	Critical	Low	High	High
7	Optimising for token cost instead of outcome cost	High	High	High	Low	High
8	Correlated failures masking as independent	Medium	High	Medium	Low	High

	review					
9	Incident reserve consumed by routine workloads	High	Medium	High	High	High
10	High task context-switching overhead	High	Medium	Medium	Medium	Medium
11	Malformed tasks starving the backlog priority	Medium	Medium	Medium	High	Medium
12	Agent micro-tasking to bypass WIP limits	Medium	Low	Medium	Low	Medium
13	Over-alerting triggering operator avoidance	High	Low	Medium	High	Medium
14	Sunk-cost fallacy delaying task abandon	Medium	Medium	Low	Low	Medium

	ment					
15	Building custom provider gateways instead of buying	Low	High	Low	High	Low

Data and research gaps

While the principles of manufacturing queueing networks, Jackson networks [3], and cloud FinOps translate well to agentic factories, empirical gaps remain regarding the true long-term service-time distributions of non-deterministic LLM generation. Furthermore, robust data is currently lacking on the long-term degradation of human attention when supervising highly autonomous software rather than manually writing it. Deep operations research is required to quantify the exact bounds of the synchronisation penalty in LLM fork-join queues where subagent tasks exhibit correlated complexity rather than independent exponential service times.[7]

Final synthesis question

What is the leanest economic operating model that allows a small organisation to run several agentic projects concurrently while controlling work in progress, protecting human attention, allocating premium and economy models rationally, limiting retries and rework, preserving incident capacity, optimising its provider portfolio, and minimising cost per accepted business outcome rather than cost per model call?

The leanest operating model is a strict pull-based Kanban architecture governed by an expected-cost routing engine and explicit capacity reservations. By enforcing rigid, platform-level Work-In-Progress (WIP) limits across projects and review stages, the system physically prevents upstream agent overproduction from overwhelming the human or integration bottlenecks, adhering strictly to Kingman's formula [2] and Little's Law.[1]

It routes tasks not by lowest initial token price, but by an Expected Total Cost equation that explicitly penalises cheaper economy models if they induce downstream rework or necessitate heavy human review.[10] It categorically rejects unlimited retries, instead mandating strategic capability escalation (e.g., moving from an economy-model-class to a premium-model-class only after a classified failure).

Human attention is actively defended through exception-driven approvals and a fixed 20% incident capacity reserve that shields operators from routine workload starvation.[15] FinOps principles ensure that provider commitments heavily favour spend-based flexibility over rigid usage-based lock-in, optimising the portfolio against rapid capability shifts.[12] Ultimately, by

attributing the full cost of all failures, retries, and human supervision to the final Accepted Outcome, this architecture mathematically aligns the platform's daily routing and admission decisions with the organisation's true business profitability.

Works cited

1. G/G/1 Queuing System and Little's Law - MATLAB & Simulink - MathWorks, accessed July 27, 2026, https://la.mathworks.com/help/simevents/ug/g-g-1-queuing-system-and-little-s-law.html
2. Kingman's formula - Wikipedia, accessed July 27, 2026, https://en.wikipedia.org/wiki/Kingman%27s_formula
3. Jackson network - Wikipedia, accessed July 27, 2026, https://en.wikipedia.org/wiki/Jackson_network
4. Networks of queues - University of Bristol, accessed July 27, 2026, https://people.maths.bris.ac.uk/~maajg/teaching/iqn/networks.pdf
5. Jackson Network Steady-State Analysis Calculator - MetricGate, accessed July 27, 2026, https://metricgate.com/docs/queueing-network-jackson/
6. Towards Cost-effective LLMs Routing with Batch Prompting - arXiv, accessed July 27, 2026, https://arxiv.org/pdf/2605.28268
7. Fork–join queue - Wikipedia, accessed July 27, 2026, https://en.wikipedia.org/wiki/Fork%E2%80%93join_queue
8. Fork-Join Queue Approximation Calculator | MetricGate, accessed July 27, 2026, https://metricgate.com/docs/fork-join-queue-approximation/
9. Response Time Approximations in Fork-Join Queues - Department of Computing, accessed July 27, 2026, http://www.doc.ic.ac.uk/~wjk/publications/lebrecht-knottenbelt-ukpew-2007.pdf
10. LLM Cost-Aware Routing: A Practical Implementation Guide | FrugalAI, accessed July 27, 2026, https://frugalai.io/blog/llm-cost-aware-routing-guide
11. Learning FOCUS: Commitment discounts - Microsoft Community Hub, accessed July 27, 2026, https://techcommunity.microsoft.com/blog/finopsblog/learning-focus-commitment-discounts/4411405
12. Commitment Discounts Overview - The FinOps Foundation, accessed July 27, 2026, https://www.finops.org/wg/commitment-based-discounts-overview/
13. A FinOps professional's guide to updated spend-based CUDs | Google Cloud Blog, accessed July 27, 2026, https://cloud.google.com/blog/topics/cost-management/a-finops-professionals-guide-to-updated-spend-based-cuds
14. 6 FinOps principles for cloud cost optimization (2026) - Flexera, accessed July 27, 2026, https://www.flexera.com/blog/finops/finops-principles/
15. SRE Capacity Planning - by Amisha Sinha - Medium, accessed July 27, 2026, https://medium.com/@amishasinha_31533/sre-capacity-planning-8f7c4b7f5cf0
16. Google SRE book - Dan Luu, accessed July 27, 2026, https://danluu.com/google-sre-book/

17. Root Cause Analysis for Probing Incident - Google SRE, accessed July 27, 2026, https://sre.google/workbook/incident-response/