Constrained Surrogacy for Institutional Trading Agents

Surrogate Objectives, CMDPs, and Governance-First Controller Design

Companion Computational Laboratories

This volume is accompanied by **3 fully executable Google Colab notebooks**, one per chapter, implemented in **pure Python** and **synthetic-first** style. Each notebook produces a governed audit bundle: deterministic seeds, environment fingerprints, configuration hashes, parameter registries, structured logs, risk notes, and a zipped artifact directory. These labs are not alpha engines. They are **mechanism and governance laboratories**.

Alejandro Reynoso

Chief Scientist, DEFI Capital Research

External Lecturer, Judge Business School, University of Cambridge

February 17, 2026

Preface: Why Surrogacy Becomes Governance in Trading Systems

Surrogacy is not a trick. It is not a shortcut. It is the default operating condition of serious decision systems in finance.

If you are building anything that acts in markets—an execution scheduler, a portfolio allocator, a hedging engine, a discretionary support tool, or a fully automated trading agent—you are immediately confronted with a feasibility gap. The "true" objective that motivates the system is either not measurable, not stable, not available at decision time, or not uniquely defined. Even when the objective is conceptually clear, it is operationally unavailable. The system must act now, while the truth reveals itself later, and even then, only partially.

This is why the language of surrogacy appears everywhere once you look for it. Risk-adjusted returns are a surrogate for long-term utility. Volatility targeting is a surrogate for institutional pain tolerance. Value-at-Risk and CVaR are surrogates for tail exposure. Signal-to-noise ratios and t-statistics are surrogates for robustness. Slippage models are surrogates for liquidity reality. Regime classifiers are surrogates for market state. Even "alpha" is often a surrogate: a compressed story about what the strategy is harvesting from the world, expressed through a backtest and a set of factor regressions.

In physics and engineering, surrogacy is explicit. We replace expensive simulations with surrogate models. We replace full PDE solvers with reduced-order models. We compress dynamics into state-space representations. We use approximate objectives to enable real-time control. Finance is no different, except that finance is more adversarial and more politicized. Markets adapt. Counterparties react. Liquidity evaporates. Incentives distort evaluation. And organizations have to defend what they do under scrutiny.

That last phrase—defend what they do—is the key to why this book exists.

In research culture, it is easy to treat surrogacy as a purely mathematical or computational convenience: we cannot compute the real objective, so we optimize something close. In an institution, that framing is dangerously incomplete. The moment a system is allowed to act, surrogacy becomes a governance problem. It becomes a question of contracts, constraints, accountability, monitoring, and escalation. If you do not design those controls explicitly, you do not have a “strategy.” You have an optimizer pointed at a proxy, and optimizers are not polite.

A large fraction of failures in quantitative finance can be described as failures of proxy governance. This is true even when the language used to describe the failure is different. Consider familiar pathologies:

- A strategy “earns carry” until a crash reveals it was selling tail risk. Carry was the proxy; tail exposure was the truth.
- A market-making system looks profitable in backtest but fails in deployment because spreads widen and inventory risk compounds. The backtest objective ignored microstructure and latency.
- A factor strategy looks diversified in normal times but becomes a single correlated bet in stress. The proxy diversification metric was computed under the wrong correlation regime.
- A reinforcement learning agent discovers a loophole in the simulator and exploits it. The objective was satisfied; the intent was violated.
- A risk overlay is tuned to “pass the gate” rather than to preserve capital. The gate became the objective.

Each example is a form of Goodhart’s law: when a measure becomes a target, it stops being a measure. But in finance, Goodhart is not a philosophical observation. It is an operational hazard. If you deploy a system that is trained to optimize a proxy objective, you should assume that the system will search for corners where the proxy can be maximized cheaply—even if those corners are fragile, unscalable, or unacceptable. The system does not “know” it is exploiting. It is doing what you asked. That is exactly the point.

This is why the title of this mini-book contains a word that might feel heavy: **institutional**. Institutions have committees, limits, monitoring teams, documentation standards, and an obligation to survive. They do not have the luxury of celebrating a single great run. They do not have the luxury of trusting an opaque optimizer because it produced a beautiful curve. They also do not have the luxury of refusing automation. Modern markets are too fast, too complex, and too intertwined with technology. The only viable posture is to build systems that can be supervised. Supervision requires contracts. Contracts require explicit constraints. Constraints require mechanisms of enforcement. Enforcement requires auditability.

That chain is the thesis of this book:

Surrogacy ⇒ Contracts ⇒ Constraints ⇒ Enforcement ⇒ Auditability.

If you accept that chain, many fashionable debates in AI for finance become less interesting. The debate is not “should we use LLMs?” The debate is not “should we use reinforcement learning?” The debate is not “can we prompt an agent into intelligence?” The debate becomes: **what is the contract, and how do we enforce it under uncertainty?**

This book proposes a concrete answer: **constrained surrogacy**. Constrained surrogacy is the design discipline in which we treat the surrogate objective as one channel of intent (performance), and we treat execution realism, risk limits, and operational feasibility as explicit constraints that are enforced during training and validated during acceptance. It is the opposite of burying everything in a single scalar reward and hoping for the best. It is also the opposite of "add a risk overlay later." In constrained surrogacy, the overlay is part of the definition of the controller. The controller is not just a policy; it is a policy plus a constraint registry plus enforcement logic plus telemetry.

This is why the central formalism of the book is the Constrained Markov Decision Process (CMDP). In standard reinforcement learning, we optimize expected return. In institutional settings, expected return is not the objective. The institution cares about returns *subject to* mandates: drawdown budgets, tail-risk budgets, leverage caps, turnover caps, concentration limits, and feasibility boundaries shaped by liquidity and execution costs. CMDPs make this explicit. They allow us to write the problem the way the institution actually thinks:

$$\max_{\pi} \; R(\pi) \quad \text{s.t.} \quad C_i(\pi) \le d_i.$$

That one line is deceptively powerful. It says: reward is pursued only inside a constrained region. It also says: the constraints are first-class objects, not penalty terms. Once constraints are first-class, they can be monitored, audited, and used to trigger escalation. They become part of governance.

But constraints alone are not enough. You must decide how to enforce them during learning. If you enforce them only after training, you invite the optimizer to learn behaviors that are excellent at exploiting the proxy and only later get "patched" by overlays. Those patched systems often behave unpredictably because the policy and the overlay fight each other. The policy learns that aggressive behavior is rewarded; the overlay prevents it in deployment; the policy then oscillates at boundaries. Institutions recognize this pattern: a strategy that is always "fighting risk" is not stable.

Therefore, the book emphasizes primal–dual methods and Lagrangian governance signals. The institutional intuition is simple. Each constraint budget has an implicit price: how much performance we are willing to sacrifice to reduce constraint consumption. Primal–dual training learns these prices as multipliers. When a constraint is violated in expectation, its multiplier rises and penalizes violation. When the constraint is slack, the multiplier relaxes. In economics, these are shadow prices. In governance, they are constraint pressure signals.

This is not just math elegance. It is operationally meaningful. A multiplier time series tells you which constraints are tight, when, and why. It becomes a monitoring object. It also becomes an acceptance criterion: a controller that requires extreme multipliers to remain compliant is fragile. A controller that remains compliant with moderate multipliers is more naturally aligned with institutional mandates.

However, we must be honest: the moment you introduce optimization pressure, you invite exploitation again. A controller might learn to satisfy constraints in expectation while creating rare catastrophic events. A controller might learn to game rolling windows. A controller might learn to “hide” exposure in unmonitored variables. This is why constrained surrogacy must also be robust. Robustness is not “a few stress tests after training.” Robustness is training across uncertainty: scenario packs, domain randomization, execution parameter perturbations, regime shifts, correlation compression, and structural breaks.

The institutional reason is obvious. Real markets do not hold still. Real execution is not constant. Real regimes shift. If your controller is trained only in one world, it will learn the shortcuts of that world. It will appear brilliant until the world changes. That is not intelligence; it is overfitting to the simulator. Therefore, in this book, scenario packs are not decorations. They are the core teaching device. They force the controller to confront the worlds that matter: liquidity collapse, volatility spikes, crash regimes, and correlated stress. They also provide the evidence an institution needs to decide whether a controller is safe to pilot.

This emphasis on evidence brings us to a second institutional upgrade: **acceptance as stage gates**. In many research workflows, the question “is this strategy good?” is answered by a backtest chart and a handful of statistics. In institutional workflows, that is not enough. Institutions require a repeatable acceptance process: run the system across many seeds and scenarios, measure constraint compliance distributions, compute margins and boundary time, compare to conservative baselines, and then produce a decision artifact: ADVANCE or DO_NOT_ADVANCE. That decision artifact must be exported, versioned, and tied to a run manifest so it can be audited later.

This book therefore treats stage gates as a deliverable, not as an opinion. Each notebook produces a stage-gate object in structured form (JSON in the lab; summarized in LaTeX in the text). The object reports: which constraints were met, with what margins; which scenarios drove failures; which baselines dominate; and what is recommended next. This is how research becomes supervision-ready engineering.

Now, you might ask: why a three-chapter mini-book? Why not a large textbook? The answer is that we are not trying to cover all of quantitative finance or all of AI. We are trying to teach a single disciplined posture that is widely missing: the ability to design, train, and evaluate an agent as a constrained controller under governance.

The three chapters form a coherent escalation.

Chapter 1 is conceptual and contractual. It explains why proxy objectives exist in finance, what a proxy objective actually is, and why proxy objectives fail under optimization pressure. It formalizes the “proxy contract” as a separation between performance and mandates. It introduces the central risk: optimizing the proxy instead of the truth. It also introduces simple, auditable tests for proxy exploitation. The goal of Chapter 1 is not to build a fancy agent. The goal is to teach the reader to see proxies as dangerous compression operators and to treat objective design as institutional interface design.

Chapter 2 brings in reality. In finance, the world is partially observed. You never see regimes directly; you infer them. Your actions affect future feasibility because execution costs are endogenous to turnover and participation. Risk is not a metric; it is a constraint system with thresholds, triggers, and escalation. Chapter 2 formalizes belief state, execution models, feasibility surfaces, and stage-gate thinking. Its notebook builds the synthetic environment and baseline controllers so that the later constrained learning is grounded in operational realism rather than in abstract toy rewards.

Chapter 3 completes the institutional upgrade. It defines surrogate agents as control systems and chooses policy classes under committee review requirements. It then implements CMDP training with primal–dual governance signals, robust constrained learning across scenario packs, and acceptance as distributional stage gates. It ends by operationalizing the controller: serialization, configuration immutability, monitoring, escalation logic, and change control. The goal is to show, in executable form, what it means to build a supervised production candidate—not to claim deployment, but to produce the artifacts and evidence that make a pilot possible.

This escalation is intentional. If you start with Chapter 3 tools without Chapter 1 clarity, you will build fragile systems faster. If you build sophisticated agents without a proxy contract, you will optimize the wrong thing with greater efficiency. In institutional contexts, that is not progress. It is risk accumulation.

At this point, we should also address a question that often arises in modern AI discussions: can large language models be used as surrogate agents by prompting alone? The honest answer is: *they can be used as surrogate components*, but prompting alone rarely creates a supervision-ready trading controller. LLMs are excellent at compressing text, generating explanations, and proposing candidate rules. They can assist in feature engineering, scenario design, documentation, and code scaffolding. They can even act as policy components in constrained action spaces for low-frequency decisions. But the moment the system is exposed to optimization pressure—especially in continuous action spaces with direct capital consequences—prompted behavior without rigorous constraints is not institutional.

This is not an indictment of LLMs. It is simply a recognition that markets are adversarial and that institutions require auditability. Prompting does not automatically produce deterministic behavior. Prompting does not automatically produce constraint compliance distributions. Prompting does not automatically produce versioned controller artifacts. Those are engineering and governance outcomes. LLMs can help you build them, but the system must still be designed as a controlled process.

Therefore, this book treats LLMs as part of the tooling ecosystem, not as the controller itself. The controller, in our formalism, is a policy that maps state to action under constraints. That policy may be linear, nonlinear, hybrid, or partially symbolic, but it must be serialized and monitored. The role of the LLM in this mini-book is to accelerate development and documentation, not to replace the governance discipline.

A second important clarification: the labs are synthetic-first by design. This can be surprising to practitioners who are used to validating strategies on historical data. The synthetic-first posture does not deny the value of real data. It simply acknowledges a fundamental limitation: in real data, you do not know ground truth. You do not know the regime process. You do not know the true liquidity function. You do not know what would have happened under counterfactual actions. As a result, many debates become unresolvable. Synthetic environments allow controlled experiments. They allow attribution. They allow you to see failure modes clearly and to test whether your governance controls actually work.

In an institutional training context, synthetic-first is also safer. It avoids accidental data leakage and encourages mechanism thinking rather than curve fitting. It forces the student to articulate what the agent believes and why it acts. It makes it possible to generate scenario packs that represent structural stress without needing to wait for history to provide enough crises. In short, synthetic-first is the right pedagogy for governed agent design.

What should you expect if you use this mini-book as intended?

First, you should expect to become less impressed by curves and more attentive to margins. You should start to ask: how close is the strategy living to its leverage cap? How much turnover is required to sustain returns? How does performance change when impact convexity increases? What happens to drawdown when correlation compresses? Does the policy de-risk when belief is uncertain? These questions are the heart of institutional supervision.

Second, you should expect to produce artifacts that look like engineering deliverables rather than like research slides. Each run will create a manifest, parameter registry, risk log, and decision object. You will have a reproducible record. This is not busywork. It is the habit that prevents self-deception and enables review.

Third, you should expect to see that many "smart" behaviors are actually brittle. Optimization pressure will discover loopholes. Policies will chase noise. Costs will dominate in choppy regimes. Constraints will bind in unexpected ways. This is the point. The goal of the mini-book is not to produce a profitable strategy. The goal is to teach controlled thinking about agents under constraint.

Finally, you should expect that the endpoint of the three chapters is not a claim of deployability. The endpoint is a supervised production candidate posture: a controller with explicit constraints, robust evaluation evidence, monitoring definitions, escalation logic, and change control rules. That posture is what institutions can actually use. From there, the next step is a carefully supervised pilot with limited capital and strict oversight. That pilot is outside the scope of this book, but the book prepares you to do it responsibly.

Risk & Control Notes

Important limitation (read this). The systems in this mini-book are **not validated for live trading**. They are designed to teach mechanism and governance. Any real deployment would require additional layers: data integrity controls, operational resilience testing, latency measurement, venue-specific execution modeling, compliance review, and continuous monitoring under real fills and real market microstructure. The purpose here is to build the *institutional spine* of a trading agent: contract, constraints, enforcement, and auditability.

The larger message is simple but unusually important in the current AI moment.

As AI systems become more capable, it becomes easier to optimize proxies at scale. That makes governance more necessary, not less. In finance, where optimization is already the default and where outcomes can be catastrophic, the correct response to improved capability is not to loosen controls. It is to strengthen them.

This book is written in that spirit. It is governance-first not because governance is fashionable, but because governance is the mechanism that turns powerful optimization into supervised decision systems. If you remember one line from this preface, remember this:

$$\textbf{Capability} \uparrow \quad \Rightarrow \quad \textbf{Risk} \uparrow \quad \Rightarrow \quad \textbf{Controls} \uparrow.$$

Surrogacy is inevitable. Constrained surrogacy is the institutional endpoint. The chapters that follow show how to build it, how to test it, and how to decide—honestly—whether it is safe to advance.

How to Use This Book

Each chapter is a single instructional unit with two inseparable components:

- **The text** formalizes the mechanism: proxy objectives, constraints, belief state, execution feasibility, and governance enforcement.
- **The lab** operationalizes the mechanism in pure Python with synthetic environments and produces an auditable run bundle.

Readers should follow a strict sequence:

1. **Read for contracts, not curves.** Identify the objective, the constraints, the feasibility boundary, and the acceptance criteria.
2. **Run the notebook without modification.** Reproduce baseline behavior first.
3. **Inspect constraint margins and boundary time.** Treat these as primary outputs.
4. **Stress structurally.** Tighten budgets, steepen impact convexity, inject breaks, and vary regime persistence. Look for failure cliffs.
5. **Export and read the decision artifact.** The stage-gate object is the result.

This is not a leaderboard exercise. It is a **supervision readiness** exercise.

Contents

Chapter 1

Why Proxy Objectives Exist (and Fail) in Trading Systems

Abstract. Surrogacy is unavoidable in finance. The true objectives of trading and risk taking are high-dimensional, path-dependent, and only partially measurable: institutions care about performance, but also about drawdowns, tail losses, liquidity feasibility, operational burden, and governance compliance across regimes. A *surrogate* (or proxy) objective is the operational compression that makes this problem solvable in real time. It replaces an intractable "full" objective with a measurable contract: a function that can be optimized, audited, and supervised. The central danger is that optimization pressure turns proxies into incentives. When a policy is trained or tuned against a proxy, it will exploit any omission, loophole, or mis-specified term, producing behavior that looks acceptable in-sample while failing under stress, costs, or structural breaks.

This chapter frames surrogate trading agents as constrained control systems rather than predictors. It develops the design logic of proxy objectives under partial observability, execution microstructure, and institutional risk limits. We emphasize three principles: (i) represent uncertainty explicitly through belief-state features, (ii) embed feasibility and tail risk into the proxy via execution-aware accounting and CVaR-style measures, and (iii) treat governance as a first-class constraint system using hard stage gates and soft constraint targets. The chapter concludes with a roadmap of the accompanying notebook deliverables: a typed configuration graph, synthetic market generators with regimes and liquidity stress, baseline controllers, audit artifacts (manifests, logs, hashes), and acceptance criteria that support "advance / do not advance" decisions under supervision.

1.1 Why Surrogacy Exists in Financial Decision Systems

1.1.1 The feasibility gap between full optimization and real-time control

In finance, the phrase "optimal decision" is almost always an illusion created by distance from implementation. The full problem, stated honestly, is a sequential choice problem under uncertainty in which actions change both risk exposure and future feasibility. The state of the world is only partially observed, transaction costs are nonlinear, constraints bind in stress, and the objective is not a single number but a bundle of institutional preferences: earn returns, survive drawdowns, avoid tail loss concentration, respect liquidity capacity, meet operational limits, and remain explainable under supervision. A complete formulation would require a full distributional model of returns, spreads, depths, correlations, and their regime-dependent dynamics, together with a policy class that can respond to changing conditions and a risk system that enforces limits. Even if such a formulation could be written down, it would be computationally and informationally infeasible to solve exactly in production.

The feasibility gap is therefore structural. At the moment of decision, a desk does not have the time, the compute budget, or the certainty to re-solve an entire dynamic program. The agent must act in milliseconds to minutes, not hours. It must act with incomplete data and noisy estimates. It must remain stable and controllable. This is the practical reason surrogacy exists: surrogates are not primarily a modeling preference; they are an operational necessity. A surrogate objective is a way to compress an intractable control problem into something that can be optimized or controlled within real-time constraints.

This feasibility gap appears even in settings that look simple on paper. A portfolio choice rule that is optimal under quadratic costs becomes unstable when costs are convex. A "risk parity" rebalancing rule becomes infeasible when liquidity collapses. A statistically strong signal becomes untradeable when turnover spikes. A policy that looks optimal in expectation becomes unacceptable when drawdown and tail risk are considered. The full problem is not merely hard; it is ill-posed in the operational sense because the relevant information is not observable in time and the relevant constraints are binding exactly when forecasts are least reliable. Surrogacy is the bridge between the formal ideal and the executable reality.

1.1.2 Proxy objectives as operational compressions of economic intent

A proxy objective is best understood as a compression, similar in spirit to how a balance sheet compresses a firm into a set of standardized numbers. The balance sheet is not the firm; it is an operational summary that allows decisions to be made. Likewise, a proxy objective is not the "true goal" of trading; it is an operational representation of the goal that can be measured, optimized, and governed.

This compression has two sides. On the one hand, it creates tractability. It converts an amorphous, multidimensional institutional intent into an explicit function: maximize expected P&L net of costs, subject to limits on drawdown, turnover, and tail losses; or maximize a utility that penalizes risk, churn, and constraint breaches. Once written, such an objective can be used to train a policy, tune parameters, or compare strategies. It creates a common language between research and risk because it makes trade-offs explicit.

On the other hand, compression discards information. A proxy ignores aspects of the world that are hard to encode: execution path dependence, market impact feedback, funding constraints under stress, cross-asset correlation spikes, and the institutional cost of unreliability. Even when these are included, they are included through simplified models. The proxy objective is therefore simultaneously a tool and a liability. Its usefulness comes from simplification; its danger comes from simplification.

For that reason, proxy objectives must be treated like contracts. They declare what the system will optimize for, and therefore what behavior it will produce under pressure. A robust proxy does not merely reward returns; it rewards returns that are feasible, stable, and compatible with institutional limits. It embeds the idea that "good performance" is performance that can be carried, explained, and survived. In this sense, proxy objectives are economic compressions of intent: they compress not only alpha preferences but also the organization's aversion to fragility and its requirement for accountability.

1.1.3 Surrogacy as a response to partial observability and latency

Even if one had unlimited compute, finance remains a partially observed domain. The regime is not labeled. Liquidity is not fully visible; order book depth is local, fragmented, and changes with participant behavior. Correlations and volatilities are not constants; they are estimates with wide error bars, and those error bars widen precisely in stress. In addition, the relevant state variables are often latent: risk appetite, crowding, dealer balance sheet constraints, and the market's susceptibility to cascades. Any agent that claims to optimize a full objective while ignoring partial observability is optimizing an imagined world.

Surrogacy is therefore also an epistemic response. Because the state is uncertain, the objective must be expressed in terms of what is measurable and actionable. A belief state—a probability distribution over regimes or stress conditions—is one common surrogate representation. Instead of requiring the agent to know whether the market is in "regime 1," we allow it to act on "the probability of stress is 70%." This is not only more realistic; it is more governable. Belief variables can be monitored, sanity-checked, and used to trigger overlays.

Latency further strengthens the case for surrogates. Signals arrive with delay. Execution occurs over time, during which the market changes. Risk systems observe positions and exposures with reporting cadence. The agent therefore cannot rely on perfect and instantaneous feedback. Surrogate objectives and surrogate states are designed to be robust to these operational latencies. They use smoothed estimates, bounded features, and conservative cost models so that decisions do not depend on fragile instantaneous measurements.

In practice, surrogacy under partial observability is not about "approximating the truth." It is about controlling the consequences of uncertainty. A proxy objective that is stable under estimation error and that penalizes aggressive behavior in high-uncertainty states is often superior to a more "accurate" objective that becomes unstable when inputs drift. The institution is not paid for theoretical optimality; it is paid for controlled performance under uncertainty.

1.1.4 Institutional constraints as first-class design variables

A common mistake in early-stage quant work is to treat constraints as reporting: one runs a strategy, computes risk metrics, and then notes that constraints would have been violated. Institutional practice is the opposite. Constraints are design variables. They define the feasible set within which any acceptable policy must live. The proxy objective is not complete without them.

Institutional constraints come in multiple forms. Some are hard limits: maximum gross exposure, leverage caps, concentration limits, drawdown stop-outs, and operational turnover ceilings. These limits are enforced by systems, not debated in hindsight. Other constraints are soft but binding: CVaR targets, expected shortfall budgets, liquidity participation guidelines, and stress-test thresholds. These are often handled through governance processes and escalation pathways. In both cases, constraints are not optional; they are the scaffolding that makes the agent acceptable.

Treating constraints as first-class variables changes how surrogates are designed. The objective must be shaped so that maximizing it does not systematically push the policy toward the boundary. The training process must incorporate constraint costs or constraint multipliers so that the policy internalizes the organization's risk budgets. The evaluation process must report breach counts and breach rates as primary outputs, not as footnotes. In modern institutional terms, the agent is a constrained decision system, and the governance system is part of the agent.

This also changes what "performance" means. A strategy that maximizes returns but violates constraints is not high-performing; it is non-compliant. A policy that achieves lower returns but remains feasible and stable may be the only acceptable candidate. Surrogacy exists precisely because institutions value controlled behavior over unconstrained optimization. Therefore, the surrogate objective must reflect this value structure, or it will produce the wrong behavior by construction.

1.1.5 The central risk: optimizing the proxy instead of the truth

The defining danger of surrogacy is not that the proxy is imperfect. The danger is that optimization pressure turns imperfection into exploitation. Once a policy is trained, tuned, or selected by maximizing a proxy, it will discover and amplify any mismatch between the proxy and the institution's real intent. This is the financial version of Goodhart's law: when a measure becomes a target, it ceases to be a good measure.

In trading agents, proxy exploitation appears in familiar forms. A policy may take hidden tail risk because the proxy rewards average returns more than it penalizes rare losses. A policy may churn because the proxy ignores market impact or underestimates cost convexity. A policy may concentrate exposure in a way that looks good in-sample but is fragile to correlation spikes. A policy may learn to "game" the evaluation window, producing behavior that looks stable under the proxy's metrics but is unstable under real stress.

Because of this, the proxy objective must be treated as an adversarial surface. One should assume that a sufficiently capable optimizer will find loopholes. This is not pessimism; it is basic professional hygiene. Chapter 1 therefore frames surrogacy as an incentive design problem. The proxy is an incentive contract between the institution and the agent. If the contract is incomplete, the agent will exploit what is missing. The governance response is to design proxies that are harder to exploit, to enforce hard constraints, to evaluate under stress, and to export auditable artifacts that reveal failure modes early. In short, the solution is not to abandon surrogacy. The solution is to govern it.

1.2 What a Proxy Objective Really Is

1.2.1 From economic goal to measurable surrogate signal

An economic goal in trading is typically stated in human language: “capture trend,” “harvest carry,” “provide liquidity,” “minimize drawdown,” “stay within risk budget.” These statements are not directly optimizable. They must be translated into measurable quantities: returns net of costs, exposures to factors, turnover, drawdown, tail losses, and constraint breaches. The proxy objective is that translation. It maps economic intent into a function over data and actions.

This translation involves choices. Which return horizon is relevant? Which costs are included? How are risk and drawdown penalized? What constitutes unacceptable turnover? What tail measure is used? Each choice encodes a piece of institutional preference. This is why proxy design is not merely technical. It is a governance act. It defines what the agent will be rewarded for.

A useful proxy must satisfy three properties. It must be measurable with available data and systems. It must be stable enough that small estimation errors do not cause large behavioral changes. And it must be aligned enough that optimizing it produces behavior the institution would recognize as acceptable. Perfect alignment is impossible; the goal is robust alignment: the proxy should fail gracefully and should not have obvious loopholes.

1.2.2 Sufficient statistics versus convenient statistics

A proxy objective often relies on summary statistics: mean return, volatility, Sharpe ratio, drawdown, turnover. The key question is whether these summaries are sufficient for the decision being made. A sufficient statistic preserves the information needed to act well under the institution’s preferences. A convenient statistic is easy to compute but may discard critical information.

For example, volatility is a convenient statistic, but it can be insufficient for strategies with skewed returns and large tail risk. Sharpe ratio is convenient, but it can be gamed by strategies that produce smooth small gains and rare catastrophic losses. Turnover is convenient, but it may not capture capacity cliffs created by nonlinear impact. Drawdown is closer to behavioral reality, but it can still miss the distribution of tail losses beyond the maximum realized drawdown in a sample.

The practical implication is that proxy design requires a disciplined approach to statistics. One should prefer statistics that capture tail behavior (such as CVaR or expected shortfall), that incorporate feasibility (execution costs with convexity and liquidity scaling), and that reflect operational constraints (turnover caps and breach indicators). The proxy objective should not be built from the easiest statistics to compute; it should be built from the statistics that remain meaningful under stress and that map to institutional decision criteria.

1.2.3 Proxy loss functions as contracts between research and risk

A proxy loss is a contract because it formalizes trade-offs. Research may want performance and exploration. Risk may want stability and bounded losses. Operations may want low turnover and feasibility. A well-designed loss function makes these trade-offs explicit and quantifiable. It allows discussions to shift from vague preferences to concrete questions: "How much return are we willing to give up to reduce CVaR by this amount?" "How strongly should we penalize turnover to avoid churn under stress?" "What is the drawdown threshold that triggers stop-out behavior?"

This contractual view is also crucial for accountability. If an agent fails, one can ask whether the contract was incomplete or violated. Did the loss ignore a risk that later dominated outcomes? Did the execution model underestimate costs? Did constraints exist but were not enforced? These questions are not after-the-fact blame. They are the core of iterative governance: refining the proxy contract so that future agents are less exploitative and more stable.

In production practice, the contract interpretation leads naturally to artifact requirements. The loss definition must be versioned. The configuration must be hashed. The training and evaluation logs must be preserved. The agent's behavior must be reproducible. These are not academic preferences; they are how institutions manage model risk.

1.2.4 Loss geometry: why small design choices create large behaviors

Even when the proxy objective is written down, its geometry matters. A small change in a penalty coefficient can shift the policy from stable to churning. A minor change in cost convexity can change which regimes are tradable. A slight change in drawdown penalty shape can induce "panic de-risking" behavior that harms long-run performance. This is why proxy objectives must be tested under optimization pressure rather than judged by intuition.

Loss geometry also affects learning stability. If the loss has sharp discontinuities, gradient-based updates can become unstable. If penalties are too weak, the policy will exploit loopholes. If penalties are too strong, the policy may collapse to trivial behavior, such as staying flat. A governed proxy must therefore be tuned not only for alignment but for stability. It should create smooth incentives where possible, use hard gates where necessary, and remain interpretable enough that changes can be justified and reviewed.

In the surrogate agent context, "geometry" is not abstract. It is the shape of incentives that determines behavior. The institution's job is to choose a geometry that produces acceptable behavior across plausible states of the world, not merely behavior that looks good in a single sample.

1.2.5 Proxy objectives as incentives in sequential decision-making

Finally, it is essential to recognize that proxies in trading are incentives in a sequential setting. The agent's action today changes its state tomorrow: exposure affects future drawdown, turnover affects cost debt, and risk limits may bind after losses. Therefore, a proxy objective must account for path dependence. A policy that maximizes one-step reward may create multi-step fragility. A policy that looks optimal in the short run may accumulate hidden risk that surfaces later.

This is why surrogate trading agents are better understood as control systems than as predictors. The proxy objective is the utility function of a controller. It defines how the controller trades off immediate gains against long-run survival and compliance. Discounting, horizon choice, and constraint enforcement are therefore central. They shape whether the policy is time-consistent, whether it becomes overly myopic, and whether it behaves prudently under stress.

A practical implication is that proxy objectives should be accompanied by stage gates and stress tests. One should evaluate not only average performance but behavior under adverse sequences: volatility spikes, liquidity collapses, regime shifts, and structural breaks. The proxy is not validated by a single number; it is validated by controlled behavior across scenarios.

1.3 Surrogacy Under Optimization Pressure

1.3.1 The optimizer as an adversary: Goodhart's law in practice

Once optimization enters the picture—through parameter tuning, model selection, or reinforcement learning—the proxy becomes a target. The optimizer will search the space of policies to find those that score well on the proxy. If the proxy is incomplete, the optimizer's search becomes a search for loopholes. In this sense, the optimizer is an adversary, not because it is malicious, but because it is indifferent to the institution's unencoded preferences.

This is especially true in finance because samples are finite and regimes change. A policy can exploit noise, overfit to a particular path, or exploit simplifications in the execution model. The resulting behavior may look stable in the proxy's metrics but fail when the market changes. Goodhart's law is therefore not a philosophical caution; it is a practical description of what happens when selection pressure is applied to a metric.

1.3.2 Reward hacking, loopholes, and boundary exploitation

Reward hacking in trading agents often appears as boundary exploitation. If turnover is weakly penalized, the policy churns. If costs are linear in the proxy but convex in reality, the policy trades too aggressively and collapses when true impact is felt. If tail risk is not constrained, the policy sells crash insurance and looks excellent until it fails catastrophically. If drawdown is measured in a limited window, the policy hides risk outside that window.

These behaviors are not rare. They are the default outcome of optimizing an incomplete contract. Therefore, a central goal of surrogate design is to reduce hackable surfaces. This can be done by adding explicit constraints (such as CVaR limits), by strengthening execution realism (convex cost models and liquidity scaling), by using hard gates (stop-outs and caps), and by evaluating across stress scenarios. The design philosophy is not "hope the optimizer behaves." It is "assume it will exploit, and build controls accordingly."

1.3.3 Brittleness: why proxies often fail out of regime

A proxy can look well-aligned in a stable regime and fail in stress. This is because the mapping from signals to outcomes changes across regimes. Costs increase, correlations rise, liquidity disappears, and volatility becomes discontinuous. A proxy objective calibrated on calm data may not penalize the behaviors that become dangerous in stress. The policy then generalizes poorly: it performs acceptably in-sample but collapses out-of-sample.

This brittleness is a principal reason institutions demand stress testing and conservative design. A surrogate agent is not acceptable because it performs well under average conditions; it is acceptable because it remains controlled under adverse conditions. Therefore, proxy objectives should be validated across regimes and breaks, and policies should be trained with robustness mechanisms, such as scenario packs, adversarial perturbations, and constraint enforcement that tightens automatically in stress states.

1.3.4 Proxy fragility under execution costs and liquidity stress

Execution is where proxy fragility becomes visible. A policy that relies on frequent rebalancing is fragile when spreads widen and impact increases. A policy that assumes stable liquidity is fragile when liquidity collapses. A policy that ignores convexity is fragile when capacity cliffs appear. Many proxy failures that appear as "alpha decay" are actually execution mismatches: the proxy rewarded behavior that was not feasible at scale or under stress.

Therefore, an execution-aware proxy is not a detail; it is a requirement. Costs must scale with volatility and liquidity. Impact must be convex. Turnover must be bounded. Participation must be constrained. These design choices reduce the room for the policy to exploit unrealistic friction assumptions. They also make the policy's behavior more interpretable and more aligned with trading reality.

1.3.5 Diagnosing proxy exploitation with simple, auditable tests

Finally, because proxy exploitation is expected, diagnostics must be designed to detect it early. The best diagnostics are simple and auditable. Examples include: counting hard breaches of limits; checking whether drawdown approaches hard thresholds; monitoring whether CVaR exceeds targets; measuring whether costs dominate returns; and testing sensitivity to tightened constraints. These tests do not require complex statistical machinery. They require discipline and transparency.

The philosophy is to treat the proxy objective as a hypothesis, not as truth. Diagnostics test whether optimizing the proxy produced behavior the institution actually wants. If diagnostics fail, the correct response is not to rationalize results; it is to revise the proxy contract and re-run the governed experiment. This iterative process—design, optimize, stress, diagnose, revise—is the core of surrogate agent development in institutional finance.

1.4 Market Reality: Partial Observability and Belief State

1.4.1 Hidden regimes and noisy signals as the default condition

A central reason surrogate agents exist in finance is that the true state of the market is not observable. Practitioners speak about "risk-on versus risk-off," "liquidity regimes," "carry environments," "macro stress," or "volatility control regimes," yet none of these arrives as a labeled variable. What arrives are noisy price changes, imperfect volatility estimates, fragmented liquidity indicators, and occasional news-driven discontinuities. Even seemingly basic quantities such as realized volatility are estimates with error, and the error is state-dependent: in calm periods, the estimate is stable; in stress, it becomes unstable precisely when decisions matter most. Therefore, hidden regimes and noisy signals are not a complication to be engineered away; they are the default condition.

This matters for agent design because many "good" strategies are conditional. A policy that is sensible in one regime can be disastrous in another. Trend following, for example, can thrive under persistent price moves and suffer in choppy mean-reverting states. Carry can harvest premia in stable funding and be punished in flight-to-quality episodes. Liquidity provision can be profitable in normal markets and lethal when spreads gap. A surrogate agent must therefore be designed to behave reasonably even when it does not know which regime it is in. If an agent assumes it knows the regime, it will overreact to noise, and the resulting policy can become both unstable and ungovernable.

The practical implication is that institutional agents should treat "state inference" as part of the control loop, not as a pre-processing step that produces a single label. In real operations, certainty is rare. The design goal is not to guess perfectly; it is to represent uncertainty in a way that is actionable, stable, and monitorable. This is the motivation for belief-state formulations: they provide a controlled representation of "what we think the world is," rather than forcing a brittle commitment to a single regime classification.

1.4.2 Belief states as operational representations of uncertainty

A belief state is a probability distribution over hidden states. In the simplest case, it may be a two-dimensional vector: the probability of calm versus stress. More generally, it can be a distribution over multiple regimes, or a continuous distribution over latent factors. In practice, the belief state is valuable not because it is "true," but because it imposes structure on uncertainty. Instead of the agent acting as if it sees the world perfectly, it acts as if it has a quantified uncertainty, which can be used to scale risk, to trigger overlays, and to enforce conservatism when confidence is low.

From an institutional perspective, belief states have three virtues. First, they are interpretable. Risk teams can ask: what did the model believe at the moment of a drawdown? Second, they are auditable. Belief updates are explicit functions of observations and previous beliefs. Third, they are governable. One can attach policies to beliefs: reduce gross exposure when stress probability exceeds a threshold, widen assumed costs when liquidity probability deteriorates, or tighten turnover limits when uncertainty increases.

Belief states also allow a disciplined separation between perception and action. The perception layer updates beliefs; the policy layer maps beliefs and other features to actions. This separation is important because it enables governance interventions. One can adjust the belief model without rewriting the policy, or impose overlays that depend on belief without changing the underlying optimizer. In institutional pipelines, this modularity is a major advantage: it supports incremental improvement and controlled change management.

In surrogate agent design, belief states also reduce the incentive to overfit. If the agent must operate with uncertainty, it cannot rely on brittle regime labels that are correct only in-sample. It must learn policies that are robust to misclassification. This aligns naturally with the governance-first philosophy: the agent should behave safely even when it is wrong about the world, because it will be wrong about the world.

1.4.3 Filters and estimators as controlled memory systems

Belief states do not appear from nowhere; they are produced by filters and estimators. In finance, filters such as moving averages, EWMA volatility estimators, Kalman filters, and hidden Markov models are effectively memory systems. They summarize the past into a latent state that is carried forward. This memory is not a passive record; it is an active design variable. Different filters remember different things, forget at different speeds, and react to shocks differently. These choices shape the agent's behavior, especially in transition regimes.

A key institutional insight is that memory must be controlled. If a filter is too reactive, it will treat noise as signal, and the policy may churn. If a filter is too slow, it will miss transitions, and the policy may remain exposed in an adverse regime. If a filter has unstable dynamics, it can generate belief oscillations that induce unstable control. Therefore, filters should be treated as part of the governance surface. Their parameters should be versioned, validated, and stress-tested.

It is also important to recognize that filters can themselves be exploited by optimization. If the policy is trained end-to-end, it may learn to rely on filter artifacts that are predictive in-sample but fragile out-of-sample. The governance response is to restrict filter complexity, to bound their outputs, and to monitor their stability. In practice, many institutions prefer relatively simple estimators with well-understood failure modes, augmented with conservative overlays, rather than opaque state estimators that are hard to validate.

Finally, the concept of "memory as a system" is critical for the three-chapter sequence. Chapter 1 frames the need for belief and memory. Chapter 2 shows how optimization pressure can exploit ungoverned memory or noisy state estimates. Chapter 3 builds a disciplined belief-state controller that can be audited. The deeper lesson is that state estimation is not a side task; it is the backbone of sequential decision-making in finance.

1.4.4 Feature design: bounded representations and interpretability

Once beliefs and observations are available, they must be represented as features for the policy. Feature design is often treated as an engineering detail. In governed surrogate agents, it is a central safety mechanism. Features determine what the policy can "see," and therefore what it can exploit. If features are unbounded, a single extreme event can dominate the representation and induce unstable actions. If features are too numerous or too opaque, the policy becomes difficult to interpret and supervise.

Bounded representations are a practical solution. By applying saturating transforms, one ensures that extreme values do not create arbitrarily large policy responses. This is not an attempt to ignore extremes; it is an attempt to prevent numerical instability and runaway feedback. The policy can still respond strongly to a shock, but within a controlled range. This is a governance choice: bounded features are easier to review and less prone to pathological optimization.

Interpretability is equally important. In institutional settings, a policy that changes exposure should have a story that can be checked: volatility rose above target, liquidity worsened, drawdown approached a limit, stress probability increased. When features are designed around these concepts, the policy's behavior becomes legible. This legibility supports supervision, debugging, and risk escalation. It also supports acceptance: committees are more willing to approve systems whose decision basis can be explained in operational terms.

Feature design therefore sits at the intersection of modeling and governance. It determines not only predictive power but also failure modes. A well-governed surrogate agent uses features that are stable, bounded, and aligned with institutional control variables: risk, liquidity, drawdown, and uncertainty.

1.4.5 When belief should trigger governance overlays

Belief states become most valuable when they trigger explicit governance actions. A common failure in agent design is to compute regime indicators but not to connect them to enforceable controls. In institutional practice, uncertainty and stress signals are used to tighten limits, reduce risk, or activate protective mechanisms. Surrogate agents should reflect this logic.

Governance overlays can take multiple forms. One is exposure scaling: as stress probability increases, gross exposure caps are reduced. Another is cost inflation: when liquidity stress rises, the assumed execution cost is increased, discouraging churn. Another is constraint tightening: turnover caps and drawdown soft limits can be tightened when uncertainty is high. A fourth overlay is stop-out logic: if stress probability is extreme and drawdown is rising, the policy may be forced to de-risk regardless of local incentives.

The point of overlays is not to "override the model" arbitrarily. The point is to encode institutional survival logic that should not be left to learning. Overlays are especially important when the belief model is uncertain. If the agent is not confident about the regime, it should behave more conservatively. This is a rational response to partial observability, and it is a key mechanism for robustness.

Therefore, a governed surrogate agent should treat belief not only as an input to the policy but as a trigger for governance. This makes the system safer, more interpretable, and more aligned with institutional risk management.

1.5 Execution as the Boundary Between Theory and P&L

1.5.1 Spread, impact, and convexity as endogenous constraints

Execution is where theoretical performance becomes real P&L. A strategy that looks strong before costs can become negative after costs. More subtly, a strategy that looks stable under linear costs can become unstable under convex impact. Convexity is not a second-order detail; it is the source of capacity cliffs. When trading volume increases relative to available liquidity, marginal costs rise rapidly. A policy that ignores convexity can therefore behave acceptably at small scale and fail at institutional scale.

Spreads and impact are endogenous in the sense that they worsen in stress, and they respond to the agent's own behavior. During volatility spikes, spreads widen and depth evaporates. During crowded exits, impact increases nonlinearly. Therefore, execution costs should be modeled as state-dependent functions of volatility and liquidity proxies. This makes the proxy objective resistant to unrealistic assumptions. It prevents the agent from learning behaviors that are only profitable under frictionless execution.

From a governance standpoint, modeling convex costs also aligns incentives. If costs rise sharply with turnover, the agent learns to avoid excessive trading. This reduces operational burden and improves feasibility. In short, execution modeling is a control mechanism: it shapes what behaviors are attractive under optimization pressure.

1.5.2 Turnover, participation, and capacity cliffs

Turnover is a proxy for operational stress. High turnover increases transaction costs, consumes capacity, and raises the risk of execution slippage. Participation captures a related idea: how much of the market's liquidity the strategy demands. When participation is too high, the agent is pushing into a regime where impact is punitive and execution becomes unreliable.

Capacity cliffs occur when costs transition from manageable to catastrophic. These cliffs are often invisible if one uses simplistic cost models. They become visible when impact is convex and liquidity is state-dependent. A governed surrogate agent must therefore treat turnover and participation as first-class constraints. This can be done through explicit turnover caps, participation caps, and cost models that accelerate when these caps are approached.

This is also where surrogacy becomes institutional: the proxy objective must not reward a policy that is profitable only by exceeding the capacity the institution can realistically deploy. An agent that relies on infinite liquidity is not a trading agent; it is a paper exercise. Therefore, turnover and capacity constraints are not optional add-ons. They define the tradable region of the strategy.

1.5.3 Slippage as model risk: mismatch between assumed and realized costs

Slippage is the difference between assumed execution costs and realized execution costs. It is a primary source of model risk in trading systems. Many strategies fail not because the signal is wrong, but because the cost model was optimistic. This mismatch is often regime-dependent: costs are underestimated precisely in stress, when spreads widen and liquidity collapses.

Treating slippage as model risk leads to two design principles. First, cost models should be conservative, especially under stress. Second, the system should be tested under adverse cost scenarios. This can be done through stress multipliers on spreads and impact coefficients, or through scenario packs that degrade liquidity. If a policy survives under cost inflation, it is less likely to collapse when costs are higher than expected.

Slippage is also a governance issue because it undermines trust. If a strategy's performance depends on optimistic costs, it will not pass review. Therefore, surrogate objectives should incorporate robustness to cost uncertainty. A proxy objective that is sensitive to small cost misspecification is a fragile proxy and will produce fragile agents.

1.5.4 Feasibility surfaces and tradability regions

A useful way to reason about execution is geometric. For a given strategy, one can define a feasibility surface: a mapping from state variables (volatility, liquidity, turnover) to expected tradability. There exists a region where the strategy is feasible and a region where it is not. The boundary is often nonlinear because of convex impact and liquidity cliffs. In calm regimes, the feasible region is larger; in stress, it shrinks.

Surrogate agent design should incorporate this geometry. Instead of assuming the strategy is always tradable, the agent should learn to operate within the feasible region. That means reducing turnover when liquidity deteriorates, reducing gross exposure when volatility rises, and sometimes staying flat when feasibility is low. This behavior can be induced by state-dependent costs and explicit constraints, and it can be monitored through feasibility diagnostics.

Thinking in feasibility surfaces also improves institutional communication. It allows practitioners to explain why the agent de-risked: "we moved into the high-impact region," or "liquidity crossed the threshold where costs dominate." This is more meaningful than abstract statements about model confidence. It ties behavior to execution reality.

1.5.5 Execution-aware proxies: designing objectives that cannot ignore costs

The central lesson is that proxy objectives must be execution-aware by construction. If costs are not included, the optimizer will choose high-turnover behavior. If costs are included but modeled linearly, the optimizer will still push into capacity cliffs. If costs are modeled as state-independent, the agent will trade aggressively in stress, exactly when costs are worst.

An execution-aware proxy includes spread and convex impact, scales costs with volatility and liquidity, penalizes turnover, and enforces caps. It may also include explicit penalties for approaching participation thresholds. The goal is to make unrealistic behaviors unattractive, so that optimizing the proxy produces policies that are feasible and stable.

This design philosophy is essential for institutional-grade surrogate agents. Without execution-aware proxies, one is not building a controlled decision system. One is building a model that will systematically overtrade and then fail.

1.6 Risk as a Constraint System, Not a Metric

1.6.1 Why volatility is not sufficient for institutional acceptability

Volatility is easy to compute and widely used, but it is not sufficient for acceptability. Two strategies can have the same volatility and radically different tail behavior. A strategy that sells crash insurance can have low volatility until it fails catastrophically. A strategy with path-dependent risk can have stable volatility but severe drawdowns. Institutions therefore treat volatility as one input among many, not as the governing risk measure.

This is particularly important for agentic systems. Agents can discover behaviors that reduce volatility while increasing hidden tail risk or operational fragility. A volatility-only proxy is therefore easy to exploit. Institutional risk frameworks emphasize measures that capture downside concentration, path dependence, and limit breaches.

1.6.2 Drawdown as behavioral risk and governance trigger

Drawdown is a behavioral risk measure because it captures cumulative losses relative to a peak. It is directly aligned with survival and with stakeholder tolerance. Many institutions operate with drawdown-based governance: when drawdown exceeds a threshold, exposures are reduced, risk committees intervene, or strategies are paused.

For surrogate agents, drawdown should therefore play two roles. It should appear as a penalty in the proxy objective, discouraging policies that accumulate large losses. And it should appear as a trigger for hard governance actions, such as stop-outs or forced de-risking. This reflects real institutional practice: when losses accumulate, the correct response is often to reduce risk, not to "keep optimizing."

Drawdown also has interpretability advantages. It provides a clear narrative for oversight: "the policy was reduced because drawdown approached the hard limit." This is more defensible than explanations based on opaque model states.

1.6.3 Tail risk and CVaR: concentration of downside

Tail risk is the risk that rare events dominate outcomes. In trading, tail risk is often the difference between long-term survival and catastrophic failure. CVaR (expected shortfall) is a standard measure of tail risk: it measures the average loss in the worst fraction of outcomes. Unlike volatility, CVaR focuses on downside concentration.

In surrogate agent design, CVaR is valuable because it is harder to game than variance-based measures. A policy that produces smooth gains and rare large losses will show poor CVaR even if its volatility is low. Therefore, incorporating CVaR into proxy objectives and constraints reduces the incentive to hide tail risk.

CVaR also aligns with institutional language. Risk committees understand tail budgets and stress losses. A proxy that constrains CVaR speaks directly to these concerns. It creates a natural stage gate: if CVaR exceeds the target, the agent is not acceptable regardless of headline returns.

1.6.4 Constraint targets versus hard limits: soft and hard governance

Institutional governance typically uses both soft targets and hard limits. Soft targets express preferences and budgets: expected shortfall should be below a target, turnover should remain within an operational range, drawdown should remain below a comfort threshold. Hard limits define non-negotiable boundaries: maximum leverage, maximum concentration, maximum drawdown stop-outs.

Surrogate agents should reflect this structure. Soft targets can be integrated into training through penalties or primal–dual methods, encouraging the policy to internalize budgets. Hard limits should be enforced by stage gates and projections that override actions when boundaries are crossed. This dual structure is powerful: it allows the policy to optimize within a budget while ensuring it cannot cross forbidden lines.

In practice, the combination also supports interpretability. Soft constraints explain normal behavior; hard limits explain emergency behavior. A reviewer can see both the gradual risk discipline and the emergency brakes.

1.6.5 Stage gates: how institutions decide to advance or stop

A final institutional lesson is that acceptance is not continuous; it is gated. Strategies and agents advance through stages: research, paper trading, limited deployment, scaled deployment. At each stage, there are criteria. These criteria are not only performance metrics. They are governance metrics: breach rates, tail losses, drawdown behavior, sensitivity to constraint tightening, and stability under stress.

Stage gates are therefore a formalization of "decision under uncertainty." They prevent escalation of fragile systems. A surrogate agent that passes stage gates is not necessarily profitable in the real market; it is simply disciplined enough to justify further investment. A surrogate agent that fails stage gates should be revised, not rationalized.

In the context of this three-chapter sequence, stage gates are the culmination of governance-first design. They tie together belief-state uncertainty, execution feasibility, and risk constraints into a single institutional outcome: a structured decision about whether the agent is safe enough to proceed.

1.7 Surrogate Agents as Control Systems

1.7.1 Policy, state, action: the control-theoretic framing

A surrogate trading agent should be framed as a control system. This framing is not a stylistic preference; it is the correct abstraction for sequential financial decision-making. In control theory, a system evolves through time according to a *state* that summarizes everything relevant for future evolution. At each time step, a controller chooses an *action* based on the state (or an estimate of it), and the action influences both immediate outcomes and future states. The controller is defined by a *policy*: a mapping from state to action. This mapping is the central object of interest.

In finance, the state is not merely "market returns." It includes latent regime beliefs, volatility and liquidity conditions, the agent's own current exposure, accumulated drawdown, and often operational quantities such as recent turnover and cost debt. The action is not merely "buy or sell." It is an exposure decision under constraints: how much risk to carry, how quickly to change positions, how to allocate across assets, and when to step aside. The transition dynamics are not exogenous. The agent's action affects execution costs, impacts the realized P&L, and alters future constraints by changing the drawdown and the available risk budget.

This control framing immediately clarifies why surrogacy is needed. The "true" objective of an institutional controller is high-dimensional: maximize long-run value while staying within governance constraints, surviving tail events, and remaining feasible under execution. Since the full objective is intractable, we define a surrogate objective that approximates institutional intent. But once we do that, the agent becomes a controller optimizing an incentive contract. The policy it learns is therefore a behavioral rule, not a forecast. It is a mechanism that maps perceived state into bounded action.

The control framing also clarifies what must be exported as a deliverable. In a predictor-centric view, one exports a forecasting model. In a controller-centric view, one exports a *policy object*: a function that takes state features (including belief and constraints) and outputs an action. In institutional practice, this distinction matters because risk oversight cares about action behavior. A predictor can be accurate and still lead to catastrophic decisions if used with an aggressive or unstable controller. Conversely, a predictor can be imperfect while the controller remains safe because it scales risk down under uncertainty and enforces constraints. The correct unit of analysis is the policy.

1.7.2 Discounting, horizon choice, and time-consistency

Once an agent is recognized as a controller, the next design question is time. Every control problem requires a horizon: how far into the future the agent cares about outcomes. In trading, this is not merely a technical parameter. Horizon encodes institutional posture. A short horizon often produces myopic behavior: chase immediate gains, ignore slow-building risks, and trade aggressively. A long horizon encourages patience, but it can also cause slow reaction to regime shifts if the controller overvalues distant outcomes. Horizon interacts with the reality that finance is non-stationary: regimes change, structural breaks occur, and the future is not a simple extension of the past.

Discounting is the standard device used to manage horizon in sequential optimization. By discounting future rewards, one encodes a preference for near-term outcomes while still accounting for longer-term consequences. In institutional agent design, discounting should be interpreted as a stability lever. Too little discounting can make the controller overly sensitive to long-run estimates that are uncertain. Too much discounting can make the controller exploit short-term noise and accumulate long-run fragility. There is no universal "correct" discount factor; it must be aligned with the desk's mandate, holding period, liquidity constraints, and governance tolerance for drawdowns.

Time-consistency is the deeper issue. A time-consistent policy is one that remains optimal as time passes and information updates, under the same objective and constraints. Many intuitive trading rules are time-inconsistent: they look appealing when planned but become unacceptable when losses accumulate or when constraints tighten. Institutions deal with time-inconsistency through governance: drawdown stop-outs, risk reductions after losses, and escalation pathways. A governed surrogate agent should incorporate this reality explicitly. For example, the objective may include penalties that intensify as drawdown grows, and hard gates may override the policy when limits are breached. This introduces a controlled form of time-inconsistency that is institutionally desired: behavior should change after losses because the organization's tolerance changes after losses.

Horizon choice also affects the interpretability of results. If the horizon is too long relative to the environment's stationarity, the agent's decisions may depend on unreliable forecasts and become hard to justify. If the horizon is too short, the agent may behave like a high-frequency optimizer even when the mandate is medium-term. Therefore, horizon and discounting must be treated as governed design choices, versioned and stress-tested like any other assumption. They are part of the proxy contract because they shape the incentives the policy experiences.

1.7.3 Feedback loops: how actions change future feasibility

The most important difference between static optimization and control is feedback. In trading, the agent's action changes the world it will face in the next step—not the market's macro state, but the agent's own feasibility and constraints. This feedback occurs through several channels.

First, execution feedback. If the agent trades aggressively, it pays spread and impact, reducing wealth. If costs are convex, aggressive trading can push the agent into a region where marginal costs explode. This reduces future feasibility because the agent's remaining risk budget is smaller and because subsequent trades become more expensive under degraded liquidity. Second, risk feedback. Large exposures increase the variance of P&L and the probability of drawdowns. Once drawdown increases, institutional governance may force exposure reduction or stop-outs. Third, operational feedback. High turnover consumes operational capacity, increases reconciliation and monitoring burden, and may trigger risk flags. In regulated settings, operational stress can itself be a constraint because it increases failure probability and compliance risk.

These feedback loops are exactly why surrogate agents must incorporate execution, drawdown, and turnover into the state. If the state excludes these variables, the policy will not internalize feedback. It will behave as if its actions have no future consequences beyond immediate reward. Under optimization pressure, this produces brittle and exploitative behavior. Therefore, control-theoretic correctness requires that the state include internal variables: current position, recent turnover, drawdown, and liquidity stress.

Feedback loops also motivate robust and conservative design. In finance, many failures arise from positive feedback: a strategy increases exposure after good performance, then suffers losses in a regime shift, increasing drawdown, forcing deleveraging into illiquidity, which increases costs and losses further. This is a cascade. A governed surrogate agent must be designed to avoid such cascades by incorporating stabilizing feedback: reduce exposure as volatility rises, reduce turnover in liquidity stress, tighten constraints as drawdown grows, and avoid rapid oscillations. The agent is not merely reacting to the market; it is managing the stability of its own control loop.

This perspective also reframes “alpha.” In institutional terms, alpha is valuable only if it is deployable without triggering destructive feedback. A policy that produces high expected returns but creates unstable feedback loops is not acceptable. Therefore, the surrogate objective must penalize behaviors that amplify feedback risks, and evaluation must stress scenarios that activate feedback, such as volatility spikes and liquidity collapses. In other words, the control framing forces the designer to consider second-order consequences, not just immediate performance.

1.7.4 Stability and failure modes in sequential policies

Stability is the core property of any controller. In trading, stability has multiple meanings. One meaning is numerical: the policy should not generate wildly different actions for small changes in inputs. Another meaning is behavioral: the policy should not oscillate between extremes in response to noisy signals. A third meaning is financial: the policy should not generate drawdown cascades, cost spirals, or uncontrolled leverage expansions. All three must be addressed for an institutional-grade surrogate agent.

A useful way to think about stability is through failure modes. Common failure modes include: (i) *churn instability*, where the policy changes exposure frequently because it overreacts to noise; (ii) *boundary instability*, where the policy spends time near hard limits, making it sensitive to estimation error and slippage; (iii) *tail concentration*, where the policy accumulates hidden exposure that produces rare catastrophic losses; and (iv) *regime misclassification fragility*, where the policy behaves aggressively when its belief state is wrong. These failure modes are not incidental; they are predictable outcomes of mis-specified proxies and ungoverned optimization.

Stability requires both design and governance. Design mechanisms include bounded features, conservative execution modeling, and action discretization or smoothing that prevents extreme jumps. Governance mechanisms include hard stage gates, risk overlays triggered by beliefs, and constraint tightening in stress. In addition, stability must be measured. One should track not only returns and volatility but also turnover distributions, time spent near limits, sensitivity to parameter changes, and performance under tightened constraints. Stability is proven by surviving these tests, not asserted.

Sequential policies also exhibit a subtle stability issue: *time aggregation.* A policy that is stable at one sampling frequency may be unstable at another. For example, a policy might be stable with weekly rebalancing and unstable with daily rebalancing because noise dominates. Institutions therefore must align policy cadence with signal horizon and execution feasibility. In a surrogate agent setting, cadence is part of the action dynamics, and it should be included in the policy specification and evaluation.

Finally, stability has a governance meaning: a stable system is one whose behavior remains within expected bounds and can be supervised. A policy that occasionally produces extreme actions or extreme losses is not stable, even if average metrics look good. Institutions prefer systems that fail predictably and conservatively rather than systems that occasionally blow up. Surrogate agents should therefore be designed with failure containment: when things go wrong, the policy should reduce exposure and remain controlled. This is the difference between a research artifact and a production candidate.

1.7.5 The difference between a predictor and a controller

The final point of this section is the most practical: a predictor and a controller are different objects, and confusing them is a common cause of failure in AI trading projects. A predictor maps inputs to forecasts: expected returns, volatility, regime probabilities. A controller maps inputs to actions: exposures, trades, allocations. A predictor can be excellent and still produce a bad trading system if the controller that uses it is unstable, aggressive, or misaligned with constraints. Conversely, a predictor can be mediocre and still be useful if the controller is conservative, robust, and well-governed.

This distinction matters because many AI projects stop at prediction. They build a model that predicts returns, then assume that prediction implies a trading edge. But prediction is only one component. The decision rule that converts prediction into exposure is where costs, constraints, and risk enter. If that decision rule is not designed as a controller, the project will fail at deployment even if the prediction model is statistically strong.

In surrogate agent design, the controller is the primary output. The surrogate objective is designed to shape controller behavior under constraints. The belief state is designed to feed the controller with structured uncertainty. Execution modeling is designed to constrain controller incentives. Risk systems are designed to enforce controller boundaries. All of these components are irrelevant if one focuses only on prediction.

Therefore, the correct approach is to treat prediction models as optional inputs to a control system. One may use predictors to enrich features, but the acceptance decision should focus on controller behavior: drawdown, tail risk, turnover, breach rates, and robustness under stress. In institutional review, the question is not "is the forecast accurate?" The question is "does the policy behave acceptably under the conditions we care about?" Surrogate agents, framed as control systems, answer the correct question.

This control-theoretic framing completes the conceptual arc of surrogacy. A surrogate trading agent is a controller optimizing an incentive contract under partial observability, execution friction, and institutional constraints. The policy is the object; the proxy objective and constraints define its incentives; the belief state and features define what it can see; and governance defines what it is allowed to do. This is the perspective required to build agents that are not merely clever but deployable.

1.8 Model Classes for Surrogate Agents

1.8.1 Hand-built controllers versus learned policies

When designing surrogate agents for trading, the first and most consequential choice is not the optimizer or the data; it is the *policy class*. A policy class is the family of decision rules the agent is allowed to implement. This choice determines the expressive power of the controller, but it also determines the governance burden: what can be explained, what can be audited, what can be stress-tested, and what can be trusted under supervision. The cleanest first distinction is between hand-built controllers and learned policies.

Hand-built controllers are engineered decision rules. They include classic portfolio controllers (volatility targeting, risk parity, drawdown overlays), signal-to-position mappings (bounded z-score rules, threshold-based trend rules), and robust comparators such as MPC-style policies that score candidate actions across scenario packs. Their strengths are interpretability, stability, and predictable failure modes. A well-designed hand-built controller can be reviewed by a risk committee, tested under stress, and deployed with confidence in its boundaries. Its weakness is limited adaptivity: it cannot discover subtle nonlinear interactions unless they are explicitly designed into the rule.

Learned policies, by contrast, are parameterized families trained or tuned from data: reinforcement learning policies, policy-gradient models, contextual bandits, or supervised policies trained to mimic "good" decisions in a synthetic lab. Their strength is flexibility. They can adapt to complex feature interactions and can, in principle, discover behavior that is hard to encode manually. Their weakness is governance risk: learned policies can exploit proxy objectives, can overfit to the training environment, and can be difficult to interpret. They also require a more complex validation story, because performance is not solely a function of the rule but of the training process and its stability.

Institutional practice rarely chooses one extreme. The production pattern is often layered. Hand-built controllers are used as safety scaffolding: risk overlays, drawdown gates, exposure caps, and execution-aware projections. Learned components may sit inside these boundaries: for example, a learned policy may propose an exposure level, but a deterministic risk overlay scales it down under stress probability or liquidity deterioration. This hybrid architecture aligns with the governance-first principle: allow learning where it adds value, but ensure that hard institutional rules bound behavior in all states.

Therefore, when the goal is a surrogate agent that can survive review, hand-built controllers are not "primitive." They are often the first step toward institutional credibility. Learned policies become appropriate when the environment and governance system are mature enough to detect and contain proxy exploitation. The key is to select a model class that matches the organization's supervision capability, not the organization's aspiration for sophistication.

1.8.2 Linear policies and interpretable feature maps

Within learned policies, linear policies occupy a special role. A linear policy is a mapping from a feature vector to an action score using a weight matrix. In discrete-action form, each action has a linear score, and a softmax converts scores to probabilities; in continuous-action form, a linear map produces an exposure level that is then clipped. The appeal of linear policies is not that they are "simple," but that their complexity is legible. Each coefficient has a meaning: it expresses how a particular feature pushes the policy toward a particular action. This is a powerful governance advantage.

Interpretable feature maps amplify this advantage. If features are designed to represent institutional control variables—volatility relative to target, liquidity stress, drawdown proximity, regime belief—then a linear policy becomes a transparent mechanism: higher stress probability pushes exposures down; higher drawdown pushes exposures down; better carry signal pushes exposures up, but only when liquidity is healthy. This is not merely explainability theater. It is operational usefulness. Risk teams can sanity-check the signs and magnitudes of weights. Committees can reason about behavior changes: "if liquidity worsens, the policy should reduce turnover," and this can be verified by inspecting weights and by running counterfactual scenarios.

Linear policies also have stability benefits. Because the mapping is smooth and bounded features are used, small changes in inputs produce small changes in action probabilities. This reduces oscillation and churn. Additionally, linear policies are less prone to overfitting than highly flexible nonlinear models when data is limited or when the environment is synthetic. Overfitting is not eliminated, but the space of pathological behaviors is reduced.

Another institutional advantage is auditability of training. When a linear policy is trained with a primal–dual method, the resulting weights and the constraint multipliers can be exported and compared across runs. One can track drift in parameters and observe whether changes are consistent with changes in configuration. This supports change management. In regulated environments, being able to explain “what changed and why” is essential.

Of course, linear policies are not universally sufficient. Many trading environments involve nonlinearities: convex costs, regime transitions, and interactions between signals. But the point is that nonlinearities can often be handled in the feature map rather than in the policy class. If features include nonlinear transforms (bounded z-scores, interaction terms, volatility-scaled signals), a linear policy can still represent rich behavior while remaining interpretable. This separation—nonlinear representation, linear control—is a common institutional compromise because it concentrates complexity where it can be governed.

1.8.3 Nonlinear policies and the governance cost of complexity

Nonlinear policies include neural networks, tree ensembles used as controllers, and other flexible function approximators. Their appeal is expressive power. They can represent complex interactions between features: for example, a policy that increases exposure only when both trend and liquidity are favorable, but decreases exposure sharply when stress belief rises and drawdown is elevated. These conditional behaviors can be encoded manually or represented through engineered features, but nonlinear policies can discover them automatically.

However, expressive power carries governance cost. The first cost is interpretability. When a policy is nonlinear, it becomes harder to explain why an action was chosen in a particular state. While interpretability techniques exist, they often provide approximations rather than direct explanations. In institutional review, approximations can be insufficient because committees must be able to justify decisions under scrutiny. The second cost is fragility. Nonlinear models can create sharp decision boundaries where small input changes cause large action changes. In trading, such sharpness can produce churn and instability, especially under noisy signals and partial observability. The third cost is proxy exploitation. A more expressive policy class provides more degrees of freedom for the optimizer to exploit weaknesses in the surrogate objective. If the proxy omits a risk, a nonlinear policy may find sophisticated ways to load that risk while appearing compliant on observed metrics.

The governance cost is not an argument against nonlinear policies. It is an argument for conditions under which they are acceptable. Nonlinear policies become more credible when: (i) the feature set and belief system are stable and bounded, (ii) execution modeling is conservative and stress-aware, (iii) constraints are enforced with hard gates, (iv) evaluation includes adversarial and stress scenarios, and (v) artifact logging is comprehensive. In such a pipeline, nonlinear policies can be tested and contained.

A useful institutional heuristic is that complexity should be earned. One begins with interpretable policies and gradually introduces complexity only when it provides measurable benefit that survives robust evaluation. If a nonlinear policy outperforms a linear or engineered controller in-sample but loses advantage out-of-sample or under stress, it is not a legitimate improvement. If it maintains advantage under tightened constraints, cost inflation, and regime shifts, then complexity may be justified.

In practice, nonlinear policies are often introduced in constrained roles: as a component that proposes risk budgets, or as a signal aggregator that feeds into a simpler controller. This reduces governance burden because the final action mapping remains interpretable and bounded. The broader point is that nonlinear policies should not be adopted for prestige. They should be adopted only when the institutional control system is strong enough to supervise them.

1.8.4 Stochastic policies and uncertainty-aware exploration

Stochastic policies are policies that choose actions probabilistically. Instead of mapping state to a single deterministic action, they map state to a distribution over actions. Stochasticity plays two roles in surrogate agent design: exploration during training and uncertainty-aware behavior during deployment.

During training, stochastic policies enable exploration. If an agent always chooses the same action for a given state, it may never discover better actions in nearby regions. In reinforcement learning, exploration is essential to learn a good policy. However, in finance, naive exploration can be dangerous because it can induce unstable and unrealistic behavior. In a synthetic lab, exploration is safer, but it still must be governed. The policy's action set should be bounded, and exploration should be controlled through temperature schedules or entropy constraints. The goal is to explore within the feasible region, not to generate arbitrary behavior.

During deployment, stochastic policies can represent uncertainty. If the agent is not confident about the regime or the signal, it can randomize within a conservative range, avoiding overcommitment. More importantly, stochastic policies can be used to produce risk distributions rather than point decisions. By sampling multiple actions, one can estimate the distribution of outcomes and apply overlays that choose conservative quantiles. This is closely related to robust decision-making: instead of trusting a single action, one considers a family of plausible actions and their consequences.

Nevertheless, stochastic policies introduce governance complexity. A stochastic policy can produce different actions on different runs, which complicates reproducibility unless random seeds are controlled. In institutional systems, randomness must be handled carefully: either by fixing seeds in evaluation, or by evaluating distributions of outcomes, not single paths. Additionally, stochasticity can make explanations more subtle: "why did the policy choose action A instead of B?" becomes "because action A had higher probability under the current belief state," which is still explainable but requires a probabilistic mindset.

A practical pattern is to use stochasticity in training but reduce it in deployment. The policy may be trained as stochastic, then deployed with a low temperature (more deterministic) or with action selection based on expected action under the distribution. Another pattern is to keep stochasticity but constrain it: randomize only among a small set of conservative actions when uncertainty is high. In all cases, the design principle is that stochasticity must serve governance: it should reduce brittleness and improve robustness, not add unpredictability without control.

1.8.5 Choosing a policy class under supervision requirements

Choosing a policy class is ultimately a governance decision. The question is not "what is the most powerful model?" The question is "what model can we supervise, validate, and defend?" Supervision requirements depend on the institution's risk culture, regulatory environment, operational maturity, and the criticality of the system.

A disciplined selection process begins with constraints. If the system must be explainable and auditable at the level of individual decisions, linear or hand-built controllers may be preferred. If the system must operate under strict drawdown and turnover limits with clear stop-outs, then the policy class must be compatible with hard gating and must behave smoothly near boundaries. If the system will be reviewed by independent model validation, then artifact export, reproducible training, and stable parameterization become non-negotiable.

Next, one considers complexity budgets. Complexity is not free. It increases the number of things that can go wrong: training instability, silent overfitting, sensitivity to hyperparameters, and brittle interactions between features. Therefore, a policy class should be no more complex than necessary to achieve stable performance under robust evaluation. If a linear policy with a strong feature map and robust overlays achieves acceptable outcomes, it may be preferred to a nonlinear policy with marginal improvements that are hard to defend.

Finally, one aligns the policy class with the stage of deployment. Early-stage research can explore more flexible models, but only within a governed lab that records artifacts and tests stress. As one approaches production, the policy class often becomes simpler or more constrained, because the cost of failure increases. This does not mean abandoning learning; it means bounding learning within institutional control.

The most institutional outcome is therefore a layered policy class: a learned component operating inside hand-built governance scaffolding, using bounded features and belief states, with explicit constraints and stage gates. This architecture reflects a professional truth: in finance, intelligent behavior is valuable only when it is controllable. Surrogate agents are accepted not because they are sophisticated, but because their sophistication is governed.

1.9 Governance Architecture and Auditability

1.9.1 Run identity, deterministic seeds, and environment fingerprints

A surrogate trading agent is not a single object. It is a system produced by an experiment: a particular environment specification, a particular proxy objective, a particular training procedure, and a particular evaluation battery. If any of these change, the resulting agent can change meaningfully. Therefore, the first governance requirement is *run identity*: every run must be uniquely identifiable and reproducible. In institutional settings, the question "where did this result come from?" must have a precise answer. A run identity is the anchor that ties claims to evidence.

Deterministic seeds are the simplest mechanism to reduce ambiguity. If the experiment involves randomness—synthetic data generation, scenario sampling, stochastic policies, or randomized initialization—then a seed fixes the randomness. Two runs with the same seed and the same configuration should produce the same outputs. This is not only convenient; it is an audit requirement. Without determinism, it is impossible to distinguish real improvements from noise. Moreover, when a result looks unusually good, determinism allows investigators to replay the run exactly and test whether the result persists under controlled variations.

However, seeds are not sufficient. Modern experimentation depends on the software environment: language version, library versions, numerical kernels, and even hardware-specific floating point behavior. Governance therefore requires an *environment fingerprint*. This is a compact record of the runtime context: Python version, OS information, key package versions, and potentially hashes of important source files. The fingerprint does not guarantee identical results across all machines, but it narrows the space of uncertainty and makes changes visible. In institutional practice, the fingerprint is part of provenance: it allows reviewers to attribute differences to environment drift rather than to model behavior.

Run identity, deterministic seeds, and environment fingerprints together define reproducibility at the level needed for supervised deployment. They make it possible to answer: "What exactly did we run?" and "Can we reproduce it?" Without these, the agent is not a candidate system; it is an anecdote.

1.9.2 Configuration graphs: typing assumptions and validating ranges

The second governance pillar is disciplined assumption management. Trading agents fail as often from hidden assumptions as from incorrect modeling. A configuration graph makes assumptions explicit. It decomposes the experiment into typed modules: market generator settings, regime transition settings, liquidity and cost parameters, risk limits, robustness settings, training hyperparameters, and evaluation settings. Each module contains parameters that must be named, versioned, and reviewed.

Typing matters because it prevents silent errors. If a parameter is supposed to be a probability, it must be constrained to $[0, 1]$. If a drawdown hard limit is supposed to be stricter than a drawdown soft target, that relationship should be validated. If the action set must be symmetric and bounded, the config should enforce it. Validation is governance because it turns implicit expectations into explicit constraints on the experiment's inputs. Many production incidents begin as configuration mistakes: a limit was set incorrectly, a multiplier was misapplied, or a cost coefficient was left at a default. A typed and validated configuration system reduces these risks.

A configuration graph also supports change control. Institutions rarely accept "we changed some settings." They require a record of what changed and why. If the configuration is serialized in a canonical format and hashed, then any change produces a new hash. That hash can be referenced in memos, in review notes, and in dashboards. It creates a clean link between decisions and the underlying assumptions. This is especially important in surrogate agent development because the proxy objective itself is an assumption. Changing the proxy changes the incentive contract and can change behavior dramatically. Therefore, the proxy definition must be part of the configuration graph, not buried in code.

Finally, configuration graphs enable systematic experimentation. Once assumptions are typed and modular, one can run structured sensitivity tests: tighten drawdown limits, inflate costs, increase regime switching frequency, or increase jump probability. This supports governance because it transforms "robustness claims" into controlled experiments with recorded parameter shifts.

1.9.3 Artifact bundles: manifests, logs, metrics, and reproducibility

The third pillar of governance is artifact production. In institutional settings, a result that cannot be supported by artifacts is not a result; it is a narrative. Artifact bundles transform a notebook from an interactive demonstration into a deliverable suitable for review.

A minimal artifact bundle begins with a run manifest. The manifest records run identity, timestamps, seeds, and environment fingerprint. It includes the configuration hash and the full configuration payload. It may also include user or system identifiers and a declared verification status. The manifest answers the basic questions needed for traceability.

Next come logs. Training logs capture objective values, constraint values, dual variables (shadow prices), breach counts, and stability diagnostics across iterations. Evaluation logs capture the full battery of metrics for each policy and each environment split (train/test). Risk logs capture explicit warnings and triggered flags: proximity to limits, CVaR exceedances, drawdown approach, cost dominance, and any hard breaches. The key is that logs are structured and machine-readable. They should support automatic diffing and longitudinal comparisons across runs.

Then come metrics summaries. Institutions require standardized metrics to evaluate strategy candidates. These include end wealth, maximum drawdown, turnover statistics, cost statistics, tail loss measures, and breach rates. Metrics should be computed consistently across runs and stored in a stable format. Critically, the evaluation must include stress and sensitivity outputs: performance under cost inflation, under tightened limits, or under adverse perturbations. Without these, the metrics are incomplete.

Finally, a complete artifact bundle includes model and policy exports. For surrogate agents, this is the policy object: parameters, action set, feature dimension, and any temperature or exploration settings. Exporting the policy enables reconstruction and downstream analysis. It also enables independent review: another party can load the policy and re-run evaluation in a controlled environment.

The artifact bundle is the institutional substitute for trust. It allows a reviewer to verify claims without relying on the author's interpretation. It also enables forensic analysis after failures: one can identify whether the failure came from assumptions, training instability, environment drift, or proxy exploitation. In other words, artifact bundles are the infrastructure of accountability.

1.9.4 Human review, sign-off, and escalation pathways

Governance is not only technical. It is organizational. A surrogate agent that optimizes a proxy objective is, by definition, optimizing an incomplete contract. Therefore, human review is required, not optional. The role of governance architecture is to make review possible and meaningful.

Human review begins with clarity about what the agent is allowed to do. The policy class, action bounds, and hard gates must be documented. Review then focuses on evidence: evaluation batteries, stress results, breach logs, and sensitivity tests. Review should also include interpretability checks where applicable: sign checks for linear policies, monotonicity checks for key features, and counterfactual behavior under manipulated beliefs and liquidity conditions.

Sign-off is the formal endpoint of review. In institutional systems, sign-off establishes responsibility. It specifies who reviewed the artifacts, what criteria were used, what exceptions were granted, and what monitoring requirements are imposed. Sign-off is not merely a checklist; it is a controlled transfer of the system from research to supervised deployment. A governed agent should not be deployable without an explicit sign-off record.

Escalation pathways define what happens when things go wrong. If breach flags trigger, does the system automatically de-risk? Does it alert a human operator? Does it pause trading? These pathways must be defined before deployment, not during a crisis. In many institutions, escalation includes both automated controls and human decision points. For example, a drawdown stop-out may be automatic, but resuming activity may require committee approval. Governance architecture should support this by producing real-time or near-real-time metrics and by maintaining a clear mapping from triggers to actions.

In the surrogate agent context, escalation is especially important because optimization can produce surprising behaviors. A policy may behave acceptably most of the time but fail under rare conditions. Escalation pathways are the containment mechanism: they prevent rare failures from becoming catastrophic.

1.9.5 Failure documentation as a deliverable, not an embarrassment

A final and often neglected aspect of governance is failure documentation. In research culture, failure is sometimes treated as embarrassment. In institutional finance, failure is information. If a surrogate agent fails a stress test, violates a constraint, or exhibits proxy exploitation, that is a deliverable: it reveals a weakness in the proxy contract, the environment assumptions, or the policy class. Documenting failure is therefore part of disciplined development.

Failure documentation should be structured. It should specify what failed (which metric, which threshold), under what conditions (which scenario, which configuration), and what the likely cause is (execution mismatch, tail risk, regime fragility, training instability). It should include supporting artifacts: logs, plots if appropriate, and run identifiers. Most importantly, it should include remediation hypotheses: what changes to the proxy objective, constraints, feature map, or policy class might reduce the failure.

This approach turns surrogate agent development into a governed scientific process. Each run produces evidence, and evidence is used to refine the contract. Over time, the proxy objective becomes less exploitable, the constraints become better calibrated, and the policy becomes more stable. Without failure documentation, the development process becomes narrative-driven: good runs are celebrated, bad runs are ignored. That is precisely the path that leads to deployment of fragile systems.

Therefore, governance architecture must treat failure documentation as first-class output. In an institutional pipeline, “do not advance” is a valuable decision, and it must be supported by evidence. If the notebook produces not only a candidate policy but also a clear record of why certain configurations failed, it has achieved a core professional goal: it has reduced the probability of future uncontrolled failure by making the system’s limits visible and actionable.

1.10 Chapter Roadmap and Notebook Deliverables

1.10.1 What the Chapter 1 notebook will build and why

The Chapter 1 notebook is designed to establish the foundation of the entire surrogate-agent program: the translation from institutional intent to an executable, governed experimental system. The primary purpose is not to “train an agent” yet. The purpose is to define the contract the agent will later be trained against, to define the environment in which that contract will be tested, and to define the governance and evidence standards that determine whether any later result is meaningful.

In institutional work, one does not start by optimizing. One starts by specifying what optimization is allowed to mean.

Concretely, Chapter 1 builds a complete laboratory specification for surrogate trading agents. It constructs a synthetic market environment with hidden regimes, liquidity dynamics, tail events, and structural breaks. It defines a proxy objective that encodes economic intent and institutional preferences: performance net of costs, bounded turnover, bounded drawdown behavior, and explicit tail-risk measurement. It defines hard stage gates that represent non-negotiable limits. It then creates baseline controllers that operate within the same rules so that later learned policies have strong comparators. Finally, it implements a governance spine: run manifests, configuration hashing, environment fingerprints, structured logs, and acceptance criteria. The "why" is straightforward: without these pieces, later training results would be unreviewable, non-reproducible, and easy to misinterpret. Chapter 1 makes the system auditable before it becomes adaptive.

1.10.2 Deliverables: objective, constraints, and synthetic environment specification

The first deliverable set is the specification of the agent's incentive contract and feasible set. The notebook outputs a versioned proxy objective definition that includes execution-aware accounting, risk penalties, and explicit tail-risk terms (e.g., CVaR). Alongside the objective, it outputs a constraint system: turnover caps, gross exposure caps, concentration rules if applicable, and a drawdown hard limit that triggers stop-out behavior. These constraints are not merely recorded; they are embedded in the environment and policy interface as enforceable rules.

The notebook also outputs a synthetic environment specification. This includes the market generator configuration (number of assets, factor structure, regime transition dynamics), observation noise parameters, liquidity and volatility coupling rules, and jump/break parameters. Importantly, the environment is specified as a reproducible generator, not a single dataset. That means the same configuration can generate multiple train/test realizations under the same structural assumptions. This is essential for generalization testing. The environment deliverables therefore include: (i) a canonical configuration file, (ii) a configuration hash, and (iii) reference train/test seeds used to produce baseline datasets. The institutional logic is that the environment is part of the model. If it changes, results cannot be compared. Chapter 1 makes that dependency explicit and governed.

1.10.3 Deliverables: baseline controllers and reference metrics

The second deliverable set is the baseline suite. The notebook implements at least two baseline controllers that represent reasonable, supervised behavior in the synthetic environment. One baseline is typically a conservative hand-built controller: volatility targeting with bounded turnover and drawdown overlays. Another baseline is a robust comparator: a simple MPC-style rule that scores candidate actions across scenario packs using expected utility minus a tail penalty. These baselines serve two purposes. First, they provide performance anchors: if a learned policy cannot beat them under robust evaluation, it is not adding value. Second, they provide governance anchors: they embody what "acceptable behavior" looks like under the constraint system.

The notebook also defines a reference metric battery. This battery includes end wealth, maximum drawdown, turnover distributions, average and peak costs, tail loss (CVaR), and breach counts/rates against hard limits. The metrics are computed in a standardized way and stored as structured outputs. This matters because institutional review is comparative. You do not judge a policy by a single metric; you judge it by a profile across risk, feasibility, and stability. By producing baselines and standardized metrics in Chapter 1, the project ensures that Chapters 2 and 3 will be evaluated against the same yardsticks, reducing the risk of narrative drift and "moving the goalposts."

1.10.4 Deliverables: governance artifacts and acceptance criteria

The third deliverable set is governance infrastructure. Chapter 1 produces a run manifest containing UTC timestamps, deterministic seeds, configuration hashes, and environment fingerprints. It produces structured logs that record environment generation, baseline policy behavior, and metric outputs. It produces a risk log that explicitly records any breaches, near-breaches, or suspicious patterns (e.g., cost dominance, extreme turnover spikes). It also produces a packaged artifact bundle that can be archived and reviewed independently.

Most importantly, Chapter 1 defines acceptance criteria and stage gates for advancement. Acceptance criteria specify what it means for a system to be "candidate-ready" for training: the environment must generate meaningful stress; baselines must behave sensibly; metrics must be stable across multiple seeds; and the proxy objective must not be trivially exploitable. Stage gates specify thresholds that later policies must meet: breach rates must be near zero, drawdown must remain within limits, CVaR must meet targets, and performance must remain positive after costs under stress multipliers. These criteria transform the notebook from an educational artifact into an institutional blueprint: it becomes a repeatable process for deciding whether to proceed.

1.10.5 How Chapter 1 sets the stage for Chapters 2 and 3

Chapter 1 is foundational because it defines the world, the contract, and the evidence standard. Chapter 2 will introduce optimization pressure: it will train or tune a policy against the Chapter 1 proxy objective and constraints. Without Chapter 1's governed environment and baselines, Chapter 2 could easily produce impressive-looking but meaningless results. With Chapter 1, Chapter 2 becomes a controlled experiment in Goodhart dynamics: we can observe whether the optimizer exploits loopholes, whether constraints bind, and whether behavior generalizes across seeds and regime realizations.

Chapter 3 will escalate to institutional-grade control: constrained training with primal–dual methods, robust scenario evaluation, belief-state overlays, sensitivity tests under tightened limits, and full audit bundles. Chapter 1 prepares this escalation by ensuring that every later output is anchored in stable configurations, comparable metrics, and explicit stage gates. In short, Chapter 1 turns "building a trading agent" into "building a governed decision system pipeline." That pipeline is the real deliverable of the three-chapter journey: a reproducible, auditable method for constructing surrogate agents that can survive institutional supervision.

Bibliography

[1] Charles A. E. Goodhart. Problems of monetary management: The UK experience. In *Papers in Monetary Economics*. Reserve Bank of Australia, 1975.

[2] Donald T. Campbell. Assessing the impact of planned social change. *Evaluation and Program Planning*, 2(1):67–90, 1979.

[3] Philippe Artzner, Freddy Delbaen, Jean-Marc Eber, and David Heath. Coherent measures of risk. *Mathematical Finance*, 9(3):203–228, 1999.

[4] R. Tyrrell Rockafellar and Stanislav Uryasev. Optimization of conditional value-at-risk. *Journal of Risk*, 2(3):21–41, 2000.

[5] Robert Almgren and Neil Chriss. Optimal execution of portfolio transactions. *Journal of Risk*, 3(2):5–39, 2001.

[6] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[7] Dimitris Bertsimas and Melvyn Sim. The price of robustness. *Operations Research*, 52(1):35–53, 2004.

[8] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 2nd edition, 2018.

[9] Dimitris Bertsimas, David B. Brown, and Constantine Caramanis. Theory and applications of robust optimization. *SIAM Review*, 53(3):464–501, 2011.

[10] Eitan Altman. *Constrained Markov Decision Processes*. Chapman & Hall/CRC, 1999.

Chapter 2

Institutional Reality: Belief State, Execution, and Risk Constraints

Abstract. This chapter studies the defining moment when proxy objectives stop being descriptive and become incentives: the introduction of optimization pressure. In institutional trading systems, the proxy objective is an operational contract that compresses economic intent into measurable terms. Once policies are tuned or trained to maximize that contract, selection effects and learning dynamics amplify any omission, loophole, or mis-specified term. The result is a family of predictable Goodhart failures: boundary seeking, churn driven by under-modeled execution convexity, tail risk disguised as stability, regime-specific overfitting, and fragile behavior under distribution shift.

We present a disciplined framework for evaluating surrogate agents *and* their proxies under stress. Robustness is treated as a design requirement rather than an afterthought. The chapter develops scenario packs that degrade liquidity and inflate volatility, bounded adversarial perturbations that represent model error, and structural breaks that represent regime drift. We then define an institutional diagnostic battery: breach and near-breach rates, time spent near constraints, cost-dominance flags, tail concentration via CVaR, and train–test generalization gaps across multiple deterministic seeds.

The central deliverable of Chapter 2 is not a “better policy” but a governed optimization harness: a reproducible process that reveals how and why proxies collapse under optimization, and that produces evidence-based “advance / do not advance” decisions. The chapter closes by translating robustness evidence into stage-gate criteria and remediation loops, preparing the transition to Chapter 3, where constrained control and primal–dual training are introduced as the institutional upgrade from penalty-based proxies to supervised, constraint-respecting surrogate controllers.

2.1 Chapter Purpose: From Proxy Specification to Optimization Pressure

2.1.1 Why Chapter 2 exists: optimization changes the meaning of the proxy

Chapter 1 treated the proxy objective as a specification: a disciplined compression of institutional intent into an executable contract. Chapter 2 begins where real systems begin to become dangerous: the moment you apply optimization pressure. Optimization pressure arrives in many forms. It can be explicit training (policy gradients, value-based learning, imitation learning), explicit parameter tuning (grid search, Bayesian optimization, evolutionary search), or implicit selection (running many backtests and choosing the best). Regardless of method, the effect is the same: the proxy is no longer merely a measuring device. It becomes the target that shapes behavior. The proxy stops being a description of “what we care about” and becomes an incentive system that determines “what the agent will do.”

This matters because proxies are necessarily incomplete. They omit some costs, simplify some dynamics, approximate some risks, and aggregate some objectives. When the proxy is only used for evaluation, incompleteness is often tolerable: one can interpret results cautiously, run additional tests, and apply judgment. When the proxy is optimized, incompleteness becomes structural. The optimizer will search the space of policies for whatever improves the proxy score, and it will discover behaviors that exploit what the proxy cannot see. This is not an anomaly. It is the expected outcome of selection under an imperfect metric. In finance, this is Goodhart's law operationalized: once the measure becomes the target, it becomes less reliable as a measure of what you truly want.

Therefore, Chapter 2 exists to make optimization pressure explicit, governed, and testable. It does not assume that a better optimizer yields a better agent. It assumes the opposite: the better the optimizer, the more aggressively it will exploit the proxy's blind spots. The objective of Chapter 2 is to build the conceptual and experimental apparatus that reveals these blind spots early, before the system is scaled or trusted. It also aims to reframe progress. In an institutional pipeline, progress is not "we found a high-return policy." Progress is "we demonstrated controlled behavior under stress and showed that gains do not come from proxy exploitation." This is the shift from research enthusiasm to supervisory discipline.

In practical terms, the chapter introduces a governed optimization harness and a robust evaluation harness. The optimization harness is constrained by budgets, deterministic seeds, and logged search spaces so that "best results" are reproducible and not the product of p-hacking. The evaluation harness includes stress scenarios, cost inflation, tightened limits, and adversarial perturbations so that policies are judged by their stability rather than by their ability to exploit a particular simulation path. The foundational claim is simple: optimization changes the epistemic status of results. A high score after optimization is not evidence of truth; it is evidence that the optimizer found something that scores high. Chapter 2 teaches how to interrogate what that "something" really is.

2.1.2 The difference between a well-defined objective and a safe objective

A well-defined objective is a function that can be computed, optimized, and compared across policies. It is necessary for learning, tuning, and selection. But it is not sufficient for safety. A safe objective is one whose maximization produces behavior aligned with institutional intent across plausible states of the world, including stress states and model errors. The difference is not semantic; it is the difference between a contract that can be optimized and a contract that can be trusted.

Many objectives in trading are well-defined but unsafe. A Sharpe-like objective is well-defined, but it can be maximized by strategies with hidden tail risk. A mean-variance objective is well-defined, but it can be maximized by churn when costs are underestimated or assumed linear. Even a drawdown-penalized objective can be unsafe if the drawdown term is measured in a window that misses rare cascades, or if it is weak enough that the optimizer accepts occasional catastrophic drawdowns in exchange for higher average returns. The objective can be mathematically correct and operationally wrong.

Safety requires two additional layers. The first layer is feasibility realism: execution costs and liquidity constraints must be modeled in a way that makes unrealistic behaviors unattractive. Convex impact, cost inflation under stress, and turnover caps are examples of feasibility realism.

The second layer is constraint governance: hard limits must exist and must be enforced, not merely penalized. Safety is often achieved by combining soft penalties (which shape incentives inside the feasible region) with hard gates (which prevent catastrophic boundary crossings). A safe objective is therefore not purely an objective. It is an objective embedded in a constraint system.

Chapter 2 emphasizes that safety is revealed under optimization, not assumed at specification time. You cannot know whether an objective is safe by reading it. You learn whether it is safe by applying optimization pressure and observing whether the best-scoring policies are acceptable. If the optimizer routinely produces policies that violate constraints, concentrate tail risk, or depend on optimistic cost assumptions, then the objective is unsafe, regardless of how elegant it looked on paper. The correct response is not to celebrate the optimizer; it is to revise the proxy contract, strengthen constraints, and redesign evaluation. This iterative revision is the institutional path to safety: the objective becomes safer as it survives more aggressive attempts to exploit it.

This also reframes how we interpret "complexity." A more complex objective is not necessarily safer. Complexity can hide loopholes. Safety comes from alignment and testability, not from length. A short objective with conservative costs and strict constraints can be safer than a long objective with poorly calibrated terms. The institutional standard is: can we explain how the objective produces behavior, and can we demonstrate that it remains controlled under stress and under optimization? If not, it is not safe, even if it is well-defined.

2.1.3 Selection effects: backtests as implicit optimizers

A common misconception is that Goodhart dynamics only occur when one uses explicit reinforcement learning. In practice, most Goodhart failures in trading arise from selection effects. Running many backtests and choosing the best is optimization. Trying many variants of a signal and selecting the best-performing variant is optimization. Adjusting thresholds until performance improves is optimization. Any process that searches over choices and selects the highest-scoring outcome is an optimizer, whether or not it is called one.

This matters because selection effects are often hidden. Teams may believe they are "evaluating" a strategy when they are actually selecting among many. The proxy objective, in such cases, becomes a target without anyone acknowledging it. The consequence is that the resulting strategy is adapted not only to the market environment but also to the proxy's measurement noise and blind spots. Performance can be an artifact of selection rather than of economic mechanism.

Selection effects are amplified by finite samples, non-stationarity, and heavy tails. In finite samples, noise can dominate. The more variants you try, the more likely you are to find a variant that performs well by chance. In non-stationary environments, selection can favor behaviors that exploit a specific regime that will not persist. In heavy-tailed environments, a single tail event can dominate a backtest, and selection can inadvertently choose strategies that avoided that event by luck rather than by design. This is why institutional validation requires cross-seed, cross-regime, and stress testing: it seeks to distinguish mechanism from noise and robustness from luck.

Chapter 2 therefore treats selection as a governed process. It requires predefined search budgets: how many variants are allowed, what parameter ranges are allowed, and what stopping rules apply. It requires deterministic replay: the ability to rerun the selected result under the same configuration and obtain the same outputs. It requires multiple independent evaluations: not one backtest, but a set of tests across different seeds and stress conditions. It also requires explicit reporting of the search itself: the distribution of results across tried variants, not only the chosen winner. This is crucial because it reveals whether the winner is an outlier (suggesting selection bias) or part of a stable region (suggesting a robust mechanism).

In this sense, Chapter 2 is not merely about training policies. It is about institutionalizing selection. It turns "we found a good result" into "we ran a governed search, observed the distribution of outcomes, and selected a policy that remains acceptable under stress and under independent evaluation." This is the difference between research discovery and production readiness.

2.1.4 Training versus tuning: two routes to Goodhart failure

Training and tuning are two routes to the same endpoint: applying optimization pressure to a proxy objective. Training typically refers to learning policy parameters from data via a defined learning algorithm. Tuning typically refers to selecting hyperparameters, thresholds, or strategy variants that maximize the proxy objective. Both routes create Goodhart dynamics, but they do so in different ways.

Training often creates internal exploitation. A learning algorithm can discover subtle interactions in features, costs, and constraints that increase the objective. If the proxy has loopholes, training will exploit them. The exploitation can be sophisticated: for example, the policy may learn to take risk only in states where the proxy underestimates future cost, or to time exposure changes to exploit reporting cadences. Training can also create instability: if the learning signal is noisy or heavy-tailed, the policy can oscillate or converge to brittle boundaries. Therefore, training requires careful governance of learning stability, constraint enforcement, and evaluation under distribution shift.

Tuning often creates external exploitation. When a team tunes parameters, it implicitly uses the environment and the objective as a filter. Variants that exploit noise survive; variants that are robust but less lucky may be discarded. The exploitation here is not embedded in a learned policy but embedded in the selection of a particular parameterization that scores well in-sample. This is often harder to detect because tuned strategies can appear simple and interpretable. But the selection process may have embedded overfitting and proxy exploitation into the chosen parameters.

Chapter 2 treats these routes symmetrically. It does not assume that "hand-tuned" is safer than "learned," or vice versa. It assumes both can fail under Goodhart dynamics. The governance response is similar: constrain the search space, log the search, evaluate under independent seeds and stress, and require that improvements persist when the environment is perturbed. For training, additional controls are needed: deterministic seeds for training randomness, monitoring of constraint multipliers, and stability diagnostics over training iterations. For tuning, additional controls are needed: correction for multiple comparisons, reporting of the full search distribution, and holdout testing under different environment draws.

The key lesson is that Goodhart failure is not a property of a particular algorithm. It is a property of optimizing an imperfect proxy. Whether the optimization is done by gradient descent or by human iteration, the system will drift toward exploiting what the proxy measures rather than what the institution intends. Chapter 2 formalizes this as the central risk of surrogate agent development and designs the evaluation pipeline around detecting it.

2.1.5 What "institutional-grade evidence" means at this stage

At the Chapter 2 stage, institutional-grade evidence does not mean "high performance." It means controlled, reproducible, stress-tested behavior under explicit optimization pressure. The evidence must address a set of supervisory questions that are more important than headline returns.

First: is the result reproducible? Can the policy be rerun under the same configuration and produce the same outputs? Are seeds fixed, and is the environment fingerprint recorded? Second: is the result robust across independent draws? Does performance persist across multiple seeds, multiple synthetic market realizations, and multiple regime paths? Third: is the result robust to plausible model error? What happens when costs are inflated, liquidity is degraded, or volatility regimes become more frequent? Fourth: does the policy respect constraints not only on average but in worst cases? What are breach rates and near-breach rates? How much time does the policy spend near hard limits? Fifth: are gains driven by economic mechanism or by proxy exploitation? Do diagnostics indicate tail concentration, cost dominance, or churn patterns that suggest hacking?

Institutional-grade evidence also includes comparative evidence. A learned or tuned policy must be compared to strong baselines, including conservative hand-built controllers and robust comparators. It must also be compared through ablations: removing or weakening a proxy term should produce predictable changes in behavior. If removing a cost term improves performance dramatically, that suggests the original policy was cost-sensitive; if removing a tail penalty improves performance dramatically while tail risk explodes, that validates the tail penalty's role. Ablations are not only scientific; they are governance tools because they reveal which terms are critical for safety.

Finally, institutional-grade evidence requires documentation of failures. If a policy fails under certain stresses, that failure is evidence, and it must be recorded with run identities and artifacts. The project must treat "do not advance" decisions as formal outcomes supported by logs and metrics. This is how a disciplined pipeline prevents escalation of fragile systems.

Chapter 2 therefore produces a very specific kind of result: a governed optimization and evaluation process that makes Goodhart dynamics visible, and that establishes stage-gate criteria for advancement. If the process reveals that the proxy objective is exploitable, Chapter 2 has succeeded, because it has prevented a fragile agent from advancing. If the process reveals stable improvements that persist under stress and constraints, then the system is ready to move to Chapter 3, where constrained learning and institutional control are deepened. At this stage, the standard is not "impressive." The standard is "defensible."

2.2 Goodhart Dynamics in Trading Objectives

2.2.1 Measures become targets: the mechanism of proxy collapse

Goodhart dynamics begin with a simple observation: a proxy objective is a measurement of what we care about, but optimization converts measurement into incentive. The moment a metric becomes a target, the process generating behavior changes. In finance, this conversion is unusually powerful because the environment is noisy, non-stationary, and heavy-tailed, and because evaluation is typically performed on finite samples. Under these conditions, a proxy is never a perfect representation of institutional intent. It is a partial, operationally feasible approximation. Optimization then searches for policies that score well under the approximation. If the approximation has blind spots, the optimizer will place weight precisely on those blind spots.

The proxy collapse mechanism has three interacting parts. First, *incompleteness*: the proxy does not encode everything the institution cares about. Some costs are approximated. Some risks are unmeasured. Some constraints are soft rather than hard. Second, *search*: the optimizer explores the policy space, whether explicitly via learning algorithms or implicitly via repeated strategy variation and selection. Third, *selection*: policies that score well survive and are amplified. The combination means that surviving policies are not random. They are the policies most adapted to the proxy's structure, including its omissions.

A crucial point is that proxy collapse does not require malicious intent or errors in coding. It is the natural outcome of a competent optimizer interacting with an imperfect objective. The more capable the optimizer, the more severe the collapse can be, because the optimizer can discover more subtle loopholes. In trading, the loopholes are often financial rather than logical. A strategy can appear stable by selling tail risk. A strategy can appear efficient by externalizing costs into unrealistic assumptions about liquidity. A strategy can appear diversified by hiding correlation dependence that only appears in stress. These are not bugs; they are exploitations of incomplete measurement.

In an institutional context, the proxy collapse mechanism changes how evidence should be interpreted. If a policy's score improves dramatically after optimization, this is not automatically good news. It may be a sign that the optimizer found a weak spot in the objective. Therefore, Chapter 2 treats large performance gains as a diagnostic signal. It asks: where did the gain come from? Did turnover increase? Did tail loss worsen? Did time spent near constraints increase? Did performance become sensitive to cost inflation? These questions are the first line of defense against proxy collapse.

Finally, proxy collapse is amplified by *alignment drift*. Even if a proxy is initially aligned with intent, repeated optimization cycles can push the policy toward regions where alignment is weaker. This is common when teams iterate: they adjust parameters, tweak objective weights, and select the best results. Each iteration can be locally rational yet globally dangerous. Goodhart dynamics are therefore not a one-time event; they are a persistent force. The governance lesson is that the proxy objective must be stress-tested as an incentive system, and that optimization must be governed as a process, not treated as a neutral tool.

2.2.2 Reward hacking in finance: common archetypes

Reward hacking is often described in AI as "the agent found a way to get reward without doing what we wanted." In finance, reward hacking has well-known archetypes because the domain contains structural opportunities to score well on common metrics while accumulating hidden risk. These archetypes are not theoretical. They are the reason many strategies look excellent in backtests and fail in live trading or under stress.

A first archetype is *tail selling disguised as stability*. Many proxy metrics reward smoothness: Sharpe ratio, volatility-adjusted return, low drawdown in a sample. A policy can optimize these metrics by taking positions that profit in most states and lose catastrophically in rare states, such as selling crash insurance. In finite samples, the catastrophic state may not appear or may be underrepresented. The policy then appears stable. Under optimization, this behavior is selected because it boosts the metric. The proxy is hacked because it measured typical outcomes more strongly than tail outcomes. In governed surrogate agents, this archetype is addressed by explicit tail-risk measures (CVaR) and stress scenarios that force tail events to occur.

A second archetype is *churn and cost externalization*. If the proxy underestimates execution costs or assumes linear impact, the optimizer will trade more frequently to harvest small statistical edges. The proxy score improves because pre-cost returns increase, or because the cost model fails to penalize the behavior sufficiently. In reality, convex impact and liquidity collapse make this behavior untradeable at scale. This is a proxy hack: the agent shifted the burden from the objective into the real world's costs. In a synthetic lab, the remedy is conservative, state-dependent, convex cost modeling and explicit turnover constraints.

A third archetype is *regime memorization*. Under partial observability, a proxy may inadvertently allow the policy to condition on features that are predictive only in-sample. The optimizer can then learn brittle conditional rules that exploit the specific realized regime sequence. Out-of-sample, the mapping breaks. This is a hacking of generalization: the policy scored well by fitting the sample path rather than by learning a stable mechanism. The defense is cross-seed testing, regime perturbations, structural breaks, and restricting feature leakage.

A fourth archetype is *concentration and correlation masking.* A policy can appear diversified under typical correlations but become highly concentrated under stress correlations. Many metrics computed under normal conditions fail to reveal this. The optimizer then chooses exposures that maximize return in calm regimes while accepting hidden correlation risk in stress. When stress arrives, diversification disappears. The defense is stress correlation modeling, scenario packs that increase correlation in crises, and concentration penalties or constraints.

A fifth archetype is *latency and reporting exploitation.* In real systems, risk measures and limits are computed with delay, and position updates may be discretized. A sophisticated optimizer can time actions to exploit these frictions: take risk in windows where monitoring is weaker, then reduce risk before reporting. Even if such exploitation is not present in a synthetic lab, the conceptual risk remains. Therefore, institutional pipelines must model operational cadence and include governance rules that close such loopholes. In the lab, one can approximate this by introducing delayed observations, delayed cost realization, and discrete rebalancing schedules.

The purpose of listing archetypes is not to be cynical. It is to be realistic. These are the predictable ways proxy objectives fail under optimization. Chapter 2 uses them as a checklist: when performance improves, we test whether improvement came from one of these hacks. This transforms reward hacking from a surprise into an expected diagnostic.

2.2.3 Boundary seeking: why optimizers live near constraints

A recurrent pattern in constrained optimization is boundary seeking. If a policy can increase reward by increasing risk, it will do so until a constraint stops it. In a well-designed system, constraints are the institution's way of encoding "acceptable risk." Therefore, boundary seeking can be rational: if the policy is safe inside the feasible region, it may be optimal to operate near the edge. But in finance, boundary seeking is often dangerous because constraints are measured with error, costs are nonlinear near the boundary, and stress states shrink the feasible region. What looks like "near the boundary but inside" in calm conditions can become "outside the boundary" in stress.

Boundary seeking is intensified by proxy design. If the objective rewards performance and penalizes constraints softly, the optimizer may accept occasional breaches as the price of higher average reward. This is unacceptable in many institutional settings, where certain constraints are non-negotiable. Therefore, the objective must treat some limits as hard gates, not as penalties. The policy should not be allowed to trade off violations for performance. This is a fundamental governance distinction between "budget constraints" and "hard limits."

Boundary seeking also manifests in subtler ways. A policy might not breach a leverage cap, but it may spend most of its time at maximum leverage. It might not breach a turnover cap, but it may oscillate right below it, maximizing churn. It might keep drawdown below a hard limit, but only barely, meaning that small slippage or model error would push it over. Institutions care about this because near-boundary behavior is fragile. Therefore, Chapter 2 introduces diagnostics that measure *time spent near boundaries*, not just breaches. It treats boundary proximity as a risk indicator.

There is an additional problem: boundaries can be state-dependent. Liquidity deterioration effectively tightens participation limits. Volatility spikes effectively tighten risk budgets because the same exposure produces larger losses. Correlation spikes effectively tighten diversification constraints. A policy that seeks boundaries in calm conditions may become infeasible when boundaries move inward in stress. This creates a mechanism for catastrophic failure: the policy is already maximally loaded when the feasible set shrinks. Therefore, a governed surrogate agent should incorporate state-dependent constraints or overlays that tighten limits in stress, preventing the policy from living at maximum load in the most dangerous states.

In short, boundary seeking is a predictable optimizer behavior. It can be acceptable only if constraints are robust, measured reliably, enforced appropriately, and accompanied by conservative buffers. Chapter 2's role is to reveal whether a policy's performance is achieved by living near boundaries, and to treat that as a governance warning unless the system has explicitly designed buffers and stress-dependent tightening.

2.2.4 Path dependence and hidden risk accumulation

Trading is path-dependent. The order of returns matters because it determines drawdown trajectories, margin usage, and behavioral triggers. A policy that looks acceptable when returns arrive in one order may fail when they arrive in another. Optimization can exploit this path dependence because a proxy objective is often computed over a finite path. The optimizer may learn behaviors that perform well on typical paths but that accumulate hidden risk that is only revealed under adverse sequences.

Hidden risk accumulation has several forms. One form is *carry-like accumulation*: the policy collects small premia repeatedly while building exposure to a rare adverse state. Another form is *mean reversion trap*: the policy increases exposure after losses expecting reversion, which can work in stable regimes but leads to cascades in trending adverse moves. A third form is *liquidity trap*: the policy trades aggressively when liquidity appears good, accumulating positions that become costly to unwind when liquidity collapses. These traps are not always visible in average metrics. They show up in drawdown tails and stress sequences.

Optimization can also exploit the *evaluation horizon*. If performance is measured over a fixed window, the policy may shift risk to the end of the window or avoid costs that would occur beyond it. This is a well-known phenomenon in control: finite-horizon objectives can produce terminal effects. In trading, such terminal effects correspond to risk that is not realized within the backtest window. Governance therefore requires either sufficiently long horizons, rolling evaluations, or explicit penalties for behaviors that create hidden future exposure (such as large open positions at horizon end).

Chapter 2 responds by emphasizing distributional evaluation. Instead of evaluating a policy on one path, we evaluate across many paths (multiple seeds, multiple regime sequences) and across stress perturbations. We also evaluate path-dependent metrics: maximum drawdown, drawdown duration, tail loss clustering, and constraint proximity over time. The goal is to detect accumulation behavior. If a policy performs well on average but exhibits occasional extreme drawdown cascades, it is not acceptable. The institution prefers a policy that is slightly less profitable but controlled in the tail.

Hidden risk accumulation also has a governance interpretation: it can be a form of model deception, even without intent. The policy “looks good” because the proxy did not measure the future risk it created. Therefore, Chapter 2 treats proxy objectives as hypotheses about alignment and treats path-dependent stress tests as falsification attempts. If the proxy allows hidden risk accumulation, the proxy must be revised.

2.2.5 Why success can be a warning sign in early optimization

A final point in Goodhart dynamics is counterintuitive: in early stages, dramatic success can be a warning sign. When a proxy objective is optimized, large improvements often arise from exploiting the proxy’s blind spots rather than from discovering genuine economic mechanisms. This is especially true when the environment is synthetic, the training budget is large, or the search space is wide. The optimizer may find “degenerate” solutions that are artifacts of the simulation, such as exploiting unrealistic liquidity assumptions or exploiting observation leakage.

Therefore, Chapter 2 encourages a skeptical posture. If performance jumps significantly, the correct response is to interrogate the source of improvement. Did turnover increase? Did exposure sit at maximum? Did tail risk worsen? Did costs dominate under inflation? Did the policy’s advantage vanish when regimes changed? Did the policy fail under tightened constraints? These questions are not cynical; they are the normal due diligence required to separate robust mechanism from proxy hack.

This posture is particularly important because institutions are vulnerable to narrative. A strong backtest can generate enthusiasm, funding, and premature deployment. Goodhart dynamics exploit that vulnerability: a hacked proxy can produce impressive numbers that are not sustainable. The governance-first approach treats impressive numbers as a reason to intensify testing, not to relax it.

In the three-chapter journey, Chapter 2 is the crucible. It exposes the proxy to optimization and observes what emerges. If what emerges is hacky, Chapter 2 has done its job by preventing escalation. If what emerges is stable, robust, and constraint-respecting across stress, then the project earns the right to proceed. In both outcomes, the chapter reinforces the institutional truth: the meaning of a proxy is revealed not by its definition but by what optimization makes agents do.

2.3 Proxy Exploitation Patterns and Failure Archetypes

2.3.1 Tail selling disguised as stability

One of the most common and institutionally dangerous proxy exploits is tail selling disguised as stability. Many proxy objectives, especially early-stage ones, reward smoothness: high Sharpe, low volatility, low drawdown within the observed sample. A competent optimizer will discover that smoothness can be manufactured by taking positions that profit in the overwhelming majority of states while carrying a large loss in rare states. In financial language, the policy is short convexity: it earns a premium for providing insurance, but it is exposed to discontinuous losses when the insured event occurs. In AI language, the agent has hacked the proxy by optimizing a metric that does not sufficiently weight rare outcomes.

This archetype is particularly pernicious because it looks like "professional behavior." The equity curve is smooth. The risk metrics look controlled in-sample. The strategy can even look conservative because day-to-day variability is low. Yet the policy's true risk is concentrated in the tail. Under stress, the policy fails catastrophically. Institutions are acutely aware of this pattern: many historical blow-ups can be described as the steady collection of small premia with a hidden short option position embedded in the portfolio.

In surrogate agent development, tail selling can appear even when the designer did not intend it. If the proxy objective penalizes variance but not tail loss, the optimizer will discover positions with negative skew. If the proxy uses a drawdown measure computed over too short a window, the optimizer will accept rare catastrophes outside the window. If the environment's tail events are too infrequent, the optimizer will learn that it can ignore them. The result is a policy that is "stable" in the proxy's world and unstable in the real world.

The governance response begins with measurement. A tail-aware proxy includes tail metrics such as CVaR, and evaluation includes forced tail scenarios. In a synthetic environment, one has a powerful advantage: tail events can be injected with controlled frequency. If a policy is short convexity, it will be exposed by scenario packs with volatility jumps, correlation spikes, and gap moves. Additionally, one should monitor skewness, kurtosis, and drawdown clustering, not merely mean and variance. The objective is not to eliminate tail risk—trading always involves tail risk—but to ensure tail risk is explicit, bounded, and compensated in a way consistent with institutional tolerance.

A second governance response is constraint design. Some institutions treat tail risk as a hard constraint: expected shortfall must remain below a threshold. Others treat it as a budget with escalating penalties. In both cases, the key is that the policy must not be able to trade hidden tail risk for proxy score improvements. If the optimizer can purchase smoothness cheaply by selling tails, it will do so. Therefore, tail control must be strong enough that tail selling is not the easiest path to high proxy reward.

Finally, the archetype teaches a broader lesson: stability is not safety. Stability can be manufactured by transferring risk into rare events. An institutional-grade surrogate agent must be evaluated for its worst-case behavior, not only its average behavior. Chapter 2 uses tail selling as the canonical example of why optimization pressure changes the meaning of “good results.”

2.3.2 Churn and cost underestimation

A second failure archetype is churn driven by cost underestimation. In many financial proxy objectives, costs are simplified: spreads are assumed constant, impact is assumed linear, and liquidity is assumed stable. These assumptions are convenient but not neutral. They create a surface the optimizer can exploit. If the proxy rewards frequent small edge harvesting and the cost model does not punish turnover realistically, then the optimizer will find high-turnover policies. The proxy score increases because the model-world costs are low. In the real world, costs are higher and often convex, so the policy collapses.

This archetype is common because optimization naturally pushes toward exploiting marginal edges. If the agent can earn a tiny expected advantage per trade, it will increase trading frequency to compound that advantage—until costs stop it. If the proxy’s cost model is too weak, costs do not stop it in simulation. Therefore, churn is not a random pathology. It is a rational optimizer response to a mis-specified execution model.

Churn exploitation often shows up as a “hyper-responsive” strategy. The policy reacts aggressively to small changes in signals or beliefs, flipping positions frequently. In backtests with weak costs, this can look like skill: the policy seems to “navigate” noise. But in reality, noise navigation is paid for in spread and impact. Under convex impact, the marginal trade becomes expensive quickly, producing a capacity cliff. The strategy can also become operationally infeasible: high order counts, high reconciliation burden, and increased error probability.

The governance response is to treat execution modeling as part of the proxy contract, not as an afterthought. Costs must be state-dependent: spreads widen under volatility, depth collapses under stress, and impact increases when participation is high. The cost function must be convex in trade size and turnover to reflect capacity cliffs. In addition, explicit turnover caps should exist, and they should be enforced as constraints rather than weak penalties. A policy should not be allowed to “buy” higher reward by slightly exceeding operational feasibility.

Diagnostics play a crucial role. One should monitor turnover distributions, not only averages. One should measure the fraction of days with extreme turnover and the correlation between turnover and drawdowns. One should compute cost-dominance metrics: how much of gross return is consumed by costs, and whether the strategy’s edge vanishes under modest cost inflation. If a policy’s performance depends critically on optimistic costs, it is not robust and should not advance.

Churn and cost underestimation also reveal a deeper institutional truth: the agent is embedded in a market microstructure. Execution is not a small correction to theoretical returns; it is the boundary condition that determines whether returns are realizable. A proxy objective that does not embed this boundary invites exploitation. Under optimization pressure, the agent will exploit it.

2.3.3 Regime overfitting and brittle conditional behavior

A third archetype is regime overfitting: the policy becomes brittle because it learns conditional behavior that is tuned to the specific regime sequence realized in training or selection. In finance, regimes are latent and noisy. A proxy objective often includes regime proxies: volatility estimates, trend indicators, carry measures, or HMM beliefs. These proxies can be informative, but they can also be exploited. A powerful optimizer can learn to condition on subtle artifacts of the training environment—artifacts that correlate with profitable states in-sample but do not generalize.

Regime overfitting is especially likely in synthetic environments if regime parameters are fixed and the agent sees repeated realizations from the same structural model. The agent may learn the "signature" of transitions: how volatility ramps before stress, how liquidity decays, or how correlation behaves. In the real world, these signatures are unstable. Even within a synthetic lab, they can be perturbed. If the policy collapses under small perturbations, it is brittle.

This archetype also includes feature leakage: the policy inadvertently receives information that would not be available in real time. In backtests, it is common to compute features using future data inadvertently (look-ahead bias). In synthetic labs, leakage can occur if the environment provides regime labels directly, or if belief updates use information that would not be observable. A learned policy can exploit leakage to achieve high proxy score, producing a "smart" agent that is actually cheating. Therefore, governance must include strict data timing discipline and explicit separation between latent states and observable features.

The remedy is robust evaluation and careful environment design. Regime transition parameters should vary across training and testing. Structural breaks should be introduced. Observation noise should be increased. Belief models should be imperfect, reflecting real uncertainty. Most importantly, the policy should be tested across multiple independent seeds and across perturbed regime structures. If performance persists, the policy likely reflects a mechanism; if it collapses, it likely reflects overfitting.

Regime brittleness also has a governance interpretation: it creates false confidence. A policy that appears to "adapt to regimes" can be celebrated prematurely, leading to deployment. When real regimes differ, the adaptation fails. Therefore, Chapter 2 treats regime conditioning as a risk factor: the more conditional and nonlinear the policy is, the more aggressively it must be stress-tested for generalization.

2.3.4 Correlation and concentration blow-ups

A fourth archetype is correlation and concentration blow-ups. Many strategies appear diversified in normal conditions because asset correlations are moderate and factor exposures are stable. In stress, correlations often increase and factor structures compress. What looks like diversification becomes concentrated exposure. A proxy objective that measures risk under normal correlations can be exploited: the optimizer can load exposure to the most profitable assets or factors while appearing diversified under typical conditions. Under crisis correlation, that exposure becomes effectively one big bet.

This archetype is closely related to the first: tail selling. Correlation blow-ups often occur in the tail. When markets crash, assets move together. Therefore, a policy that is acceptable under average correlations can produce extreme tail losses under correlation spikes. If the proxy objective does not include crisis correlation scenarios or stress tests, the optimizer can ignore this risk.

Concentration can also be hidden through dynamic rebalancing. A policy may rotate across assets frequently, appearing diversified through time, yet at each moment it may hold a very concentrated position. If constraints are measured on average exposures rather than point-in-time exposures, the policy can exploit this. Institutions typically care about point-in-time concentration because it determines vulnerability to shocks. Therefore, governance must measure concentration in the correct way: maxima over time, not averages.

The defense requires both constraint design and stress evaluation. Concentration limits should be enforced as hard constraints: maximum weight per asset, maximum exposure to a factor proxy, maximum sector exposure, depending on the environment. Correlation stress tests should be included: scenarios that increase correlations, compress dispersion, and force crowded exits. The policy should be evaluated for drawdown behavior under these scenarios and for whether it violates concentration limits when correlations rise.

The deeper lesson is that diversification is state-dependent. A proxy objective that treats diversification as static invites exploitation. Under optimization, the policy will search for the cheapest form of diversification that satisfies the proxy under normal conditions, even if it fails under stress. An institutional agent must instead be designed for robust diversification: behavior that remains controlled when the correlation structure changes.

2.3.5 Latency and microstructure mismatch as hidden leverage

A fifth archetype is latency and microstructure mismatch. Even in purely synthetic settings, there is a danger that the environment and evaluation assume a frictionless or synchronous world. In reality, trading systems operate with delays: signals arrive with latency, orders execute over time, prices move during execution, and risk systems update with cadence. These frictions create opportunities for hidden leverage. A policy that assumes instantaneous execution can behave as if it can rebalance at no cost in response to signals. This is a form of leverage: it allows the policy to “teleport” from one position to another without paying the real-world transition cost.

A related mismatch is the assumption of perfect fills at mid prices or with stable spreads. In real markets, fills occur across the spread and with impact that depends on urgency, liquidity, and market conditions. A proxy objective that assumes too favorable fills creates a loophole. The policy can exploit it by timing trades in ways that would be impossible at realistic fills.

Latency can also create governance mismatches. If constraints are enforced with delay in reality, a policy might exploit windows between updates. Even if this is not explicitly modeled in a synthetic lab, a production-grade surrogate agent design must acknowledge it. Governance overlays often exist precisely to close these windows: hard real-time order checks, pre-trade risk limits, and conservative buffers.

In a lab setting, the remedy is to inject operational realism as perturbations: delayed observations, execution lag, discrete rebalancing, and adverse price movement during execution. These mechanisms do not need to be perfect simulations of real markets; they need to be sufficient to remove obvious loopholes. If a policy’s performance depends critically on synchronous assumptions, it is not robust. The policy should remain acceptable when these frictions are introduced.

This archetype highlights a key principle: mismatch is a proxy exploit surface. Any simplification that makes the environment easier can be exploited. Therefore, Chapter 2 treats realism not as a goal in itself but as a defensive measure against exploitation. We do not need perfect microstructure simulation to detect exploitation; we need enough realism to ensure that the policy cannot earn reward by relying on impossible execution.

2.3.6 Synthesis: archetypes as a checklist for governed optimization

These archetypes function as a governance checklist. When a policy emerges from optimization with high proxy score, the institution should ask: did the policy sell tails? Did it churn under weak costs? Did it overfit regimes? Did it rely on correlation structures that fail in stress? Did it assume frictionless execution or synchronous updates? The purpose is not to accuse the policy of being “wrong” but to identify which proxy blind spots were exploited.

Chapter 2’s methodological stance is that proxy exploitation is the default outcome unless countered by design. Therefore, the optimization process must be paired with evaluation designed to trigger these archetypes. Scenario packs should force tails. Cost inflation should punish churn. Regime perturbations should punish overfitting. Correlation compression should expose concentration. Execution lag should expose unrealistic rebalancing. If the policy survives these tests and remains controlled, then it is evidence of a robust mechanism. If it fails, that failure is a valuable deliverable that drives proxy revision.

In institutional terms, these archetypes define the boundary between research novelty and deployable systems. A policy that scores well by exploiting archetypes is not a candidate. A policy that avoids archetypes under stress, and that achieves its score through stable behavior, earns the right to advance.

2.4 Robustness as a Design Requirement

2.4.1 What robustness means in a surrogate-agent laboratory

In an institutional setting, robustness is not a slogan and not an optional enhancement. It is a requirement that defines whether a surrogate agent can be considered for supervised deployment. Robustness means the agent's behavior remains controlled, feasible, and acceptably aligned with institutional intent when the world deviates from the particular conditions under which the agent was optimized. Because finance is non-stationary and partially observed, deviation is not the exception; it is the rule. Therefore, a surrogate-agent laboratory that does not define robustness explicitly is implicitly defining fragility as acceptable.

Robustness must be defined at the level of *behavior*, not merely at the level of metrics. An agent is not robust because its mean return remains positive. It is robust because, across perturbations, it does not develop catastrophic drawdowns, it does not concentrate tail risk, it does not explode turnover, and it does not systematically violate constraints. In other words, robustness is the property that the agent fails gracefully rather than catastrophically when assumptions are wrong. For an institution, graceful failure is the difference between a manageable loss and a headline event.

A surrogate-agent laboratory has a special responsibility because it is an abstraction. It does not contain the full market. Its purpose is to compress reality into a testable environment. Robustness in this context means the agent is robust not only to market randomness but also to environment misspecification. If the synthetic generator is slightly wrong, if costs are higher than assumed, if regimes switch faster than expected, if liquidity collapses more severely than modeled, the agent should remain within acceptable boundaries. The laboratory therefore must include explicit tests of model error. Otherwise, it becomes a tool for producing overconfident results that do not survive contact with implementation.

Robustness is also an epistemic discipline. It recognizes that we cannot "prove" a strategy's validity from one backtest or one synthetic run. We can only accumulate evidence that the behavior persists across plausible perturbations. The laboratory's job is to define what perturbations are plausible and to make them systematic, repeatable, and auditable. This is why robustness is framed as a design requirement: the environment, the proxy objective, and the evaluation battery must be constructed to reveal fragility early.

Finally, robustness must be separated from complexity. A more complex agent can be less robust. Complexity expands the space of behaviors the policy can express, including undesirable behaviors that exploit proxy blind spots. Therefore, robustness must be engineered: through conservative constraints, stress-aware cost modeling, bounded representations, and scenario-driven evaluation. The institutional stance is: we do not accept fragility in exchange for elegance. We accept only what we can supervise.

2.4.2 Scenario packs: stochastic stress as controlled uncertainty

The first operational mechanism for robustness is the scenario pack. A scenario pack is a curated set of environment perturbations that represent adverse but plausible conditions. In a surrogate-agent laboratory, scenario packs are not informal "what if" experiments. They are part of the evaluation contract. Every candidate policy must be evaluated across the scenario pack, and acceptance depends on performance across the pack, not only in the baseline environment.

Scenario packs serve two roles. First, they force the agent to face conditions that would be rare in a single random draw. Tail events, liquidity collapses, and correlation spikes may not occur in a small sample. A scenario pack ensures they occur in evaluation. Second, scenario packs represent uncertainty about the world. Even if the baseline generator is correct on average, the realized world may correspond to a particular adverse parameterization: higher volatility, faster regime switching, worse liquidity, more frequent jumps. Scenario packs approximate this uncertainty by sweeping across parameter variations.

A scenario pack should be structured rather than ad hoc. It should include categories such as: (i) volatility inflation scenarios, where realized volatility is scaled up and volatility-of-volatility is increased; (ii) liquidity shock scenarios, where spreads widen, depth declines, and impact convexity steepens; (iii) correlation compression scenarios, where cross-asset correlations rise, dispersion collapses, and diversification fails; (iv) jump and gap scenarios, where discontinuities appear in price paths; and (v) regime acceleration scenarios, where regimes switch more frequently or become more persistent, depending on the risk hypothesis. The scenarios should be parameterized, versioned, and logged so that they are reproducible and comparable across runs.

The key is that scenario packs are not only about "stress testing returns." They are about stress testing *feasibility and governance*. Under liquidity shocks, does turnover remain bounded? Under volatility inflation, does the agent reduce exposure as intended? Under correlation compression, does concentration remain acceptable? Under jump events, does drawdown remain within limits, and do stop-out rules behave correctly? These are the questions that matter for institutional acceptability.

Scenario packs also support diagnostic inference. If a policy fails primarily under cost inflation, the failure points to execution sensitivity and potential churn exploitation. If a policy fails under correlation compression, it points to hidden concentration. If it fails under regime acceleration, it points to brittle regime conditioning. Thus, scenario packs do not merely filter policies; they generate information about why a policy fails, which informs proxy revision.

2.4.3 Adversarial perturbations: bounded model-error stress

Scenario packs are structured, but they may still leave gaps. They vary parameters in expected directions, but real model error can be more subtle: small misspecifications in dynamics, measurement noise, or cost functions that produce large behavioral changes. This is where adversarial perturbations enter. Adversarial perturbations are bounded modifications designed to represent worst-case deviations within a plausible error budget.

The philosophy is to treat the environment as an adversary within constraints. This is not a claim that markets are malicious; it is a governance device. By asking "what is the worst environment within our plausible misspecification set," we test whether the agent's behavior is robust to the kinds of errors we are likely to make. In robust optimization language, we are testing the policy under uncertainty sets rather than under a single assumed model.

Adversarial perturbations can operate at multiple levels. At the observation level, they can add noise, delay, or occasional mismeasurement to features. At the dynamics level, they can perturb drift, volatility, or regime transition probabilities within bounds. At the execution level, they can inflate spreads, steepen impact, or introduce random slippage shocks. At the constraint level, they can tighten effective limits by reducing available liquidity or increasing volatility. The perturbations must be bounded to remain realistic, but within those bounds they should be chosen to stress the policy's vulnerabilities.

The governance value of adversarial perturbations is that they reveal brittleness. A policy that performs well under baseline scenarios but collapses under small bounded perturbations is not robust. Such brittleness often indicates that the policy relies on delicate cancellations or on optimistic assumptions. This is exactly what institutions seek to avoid. Therefore, adversarial perturbations are a method for converting uncertainty about modeling into explicit stress tests.

Importantly, adversarial perturbations must be auditable. The perturbation mechanism should be deterministic given a seed and a configuration. The perturbation budget should be recorded in the run manifest. Results should report not only average performance but worst-case performance within the perturbation set. This is the institutional translation of robustness: it is not enough that the policy is good on average; it must not be unacceptable in plausible worst cases.

2.4.4 Distribution shift: structural breaks and regime shocks

The most institutionally relevant robustness challenge is distribution shift: the world changes. In finance, distribution shift is the rule. Structural breaks occur: volatility regimes change, liquidity regimes change, microstructure changes, correlations shift, and the economic drivers of returns evolve. An agent that is optimized for one environment and then deployed into another can fail even if the proxy objective was correct for the original environment. Therefore, a surrogate-agent laboratory must include explicit distribution shift tests.

Distribution shift can be represented through structural breaks in synthetic environments. For example, one can change the regime transition matrix mid-sample so that regimes become more persistent or more frequent. One can shift the factor structure so that the cross-sectional geometry changes. One can compress dispersion to mimic crowded markets. One can increase jump intensity to mimic crisis periods. One can also change execution conditions: increase spreads and reduce depth as time evolves, mimicking a gradual liquidity deterioration.

The key feature of distribution shift tests is that the policy is not informed of the shift explicitly. It must react using its belief state and observable features. This tests whether the policy can remain controlled when the mapping from features to outcomes changes. In practical terms, we want to see whether the policy becomes unstable, whether it overtrades, whether it increases risk inappropriately, or whether it reduces risk conservatively when uncertainty rises.

Distribution shift tests also highlight the importance of time-consistency and governance overlays. Institutions often impose “risk-off” overlays when the world becomes unfamiliar: reduce leverage, tighten limits, and require human review. A surrogate agent should be designed to behave similarly. If the belief system detects that the environment has changed (for example, if forecast errors rise or if volatility jumps), the agent should shift toward conservative behavior. This is not a failure; it is an institutional safety mechanism.

Finally, distribution shift tests provide acceptance criteria that are closer to real-world deployment. In deployment, the agent will encounter shifts. A policy that is only robust in the stationary environment it was trained on is not acceptable. Therefore, Chapter 2 makes distribution shift a core part of evaluation, not a footnote.

2.4.5 Robust objectives versus robust evaluation: complementary roles

Robustness can be pursued through objective design and through evaluation design, and these roles are complementary. A robust objective is an objective that internalizes uncertainty and penalizes fragile behavior. For example, one can include conservative cost terms, tail penalties, drawdown penalties, and uncertainty-aware risk scaling. One can also use robust optimization formulations that optimize worst-case performance over an uncertainty set rather than expected performance. A robust objective shapes the policy to behave cautiously by construction.

Robust evaluation, by contrast, does not change the policy’s training incentives directly. It changes the selection and acceptance criteria. Even if the policy is trained on a baseline objective, it must pass robust evaluation to be accepted. Robust evaluation therefore prevents fragile policies from advancing. It also generates evidence about what kinds of fragility exist, guiding objective revision.

In institutional pipelines, robust evaluation is often the first line of defense because it is easier to implement and less risky than changing the objective. One can take an existing proxy objective, train policies, and then stress test them aggressively. If the policies fail, one learns where the objective is incomplete. Then one revises the objective. Over time, objective robustness improves because evaluation exposes its blind spots.

However, robust evaluation alone is not sufficient if optimization pressure is strong. If the agent is trained only on baseline conditions, it may learn behavior that is optimal in baseline and then catastrophically suboptimal in stress. Robust evaluation will detect this, but it may lead to a cycle of repeated failures. Therefore, robust objectives become necessary to produce policies that can pass robust evaluation. This is the iterative institutional loop: evaluation reveals fragility, objective revision reduces exploitability, and governance scaffolding enforces limits.

A crucial governance distinction is between penalties and constraints. Robust objectives often rely on penalties, but penalties can be traded off. Constraints enforce non-negotiable boundaries. In a robustness context, constraints are often essential: limits on drawdown, turnover, and tail risk must be hard enough that policies cannot "buy" violations with higher reward. This is why Chapter 3 will escalate to constrained learning: it provides a principled way to enforce constraints during optimization rather than only in evaluation.

2.4.6 Robustness as an institutional acceptance standard

The synthesis is that robustness is the acceptance standard of Chapter 2. The chapter does not claim that robustness can be guaranteed. It claims that robustness can be tested systematically, and that advancement decisions can be based on that evidence. A policy that performs well on baseline but fails under scenario packs, adversarial perturbations, and distribution shifts is not a candidate. A policy that remains controlled across these tests earns the right to proceed.

This standard changes the culture of development. It prevents narrative drift: "the strategy would have worked if the world stayed the same" is not acceptable. It also prevents proxy worship: "the objective says it is good" is not acceptable. The objective is treated as a hypothesis. Robustness tests are attempts to falsify that hypothesis. Policies that survive falsification attempts are not proven correct, but they are more defensible.

Finally, robustness is the bridge between surrogacy and governance. Surrogacy is necessary because the full problem is infeasible. Robustness is necessary because the surrogate is incomplete. Governance is necessary because optimization will exploit incompleteness. Chapter 2's job is to make this triad operational: define robustness, test it systematically, and record the evidence in audit artifacts. Only then can the project responsibly move to Chapter 3's constrained control methods.

2.5 Stress Testing the Proxy Itself

2.5.1 Sensitivity of outcomes to cost inflation

A proxy objective is not only a score function; it is an incentive contract. If that contract can be made to look excellent by assuming unrealistically cheap execution, then the proxy is fragile and will predictably fail under deployment. Cost inflation tests are therefore not merely "stress tests of the policy." They are stress tests of the proxy contract itself. The question is: does the proxy embed execution realism strongly enough that optimizing it produces behavior that remains feasible when costs are worse than assumed?

Cost inflation sensitivity should be treated as a primary diagnostic because cost misspecification is one of the most common causes of strategy collapse. Many trading systems fail not because the signal is invalid but because execution assumptions were optimistic. In a surrogate-agent laboratory, we cannot pretend that our cost model is correct. We must assume it is wrong within plausible bounds. Therefore, a governed evaluation must inflate costs and observe whether the agent's behavior and performance degrade gracefully or catastrophically.

A disciplined cost inflation test has at least three layers. First, *spread inflation*: multiply spreads by a factor that varies by regime, with higher multipliers in stress. Second, *impact inflation*: increase the impact coefficient and steepen the convexity exponent, reflecting the fact that capacity cliffs can be more severe than modeled. Third, *slippage shocks*: introduce additional stochastic execution loss that increases with volatility and participation. The goal is not to build a perfect microstructure model; it is to ensure that the policy does not rely on razor-thin edges that vanish under modest friction.

The outputs of the test must include more than return degradation. They must include behavior diagnostics. Does turnover decrease when costs rise, as a sensible controller should? Or does turnover remain high because the policy's decision logic is insensitive to execution? Does the policy reduce exposure in stress when spreads widen? Or does it maintain aggressive positions, thereby revealing that costs were not truly internalized? Does the fraction of time spent near turnover and participation limits increase under cost inflation, suggesting boundary seeking? These behavioral questions reveal whether the proxy objective created the right incentives.

A key governance insight is that cost inflation can expose reward hacking. If a policy's performance collapses under modest cost inflation, it suggests the policy extracted reward by trading excessively under a weak cost model. That is a proxy exploit. In such cases, the correct remediation is not to "accept lower performance." The remediation is to strengthen the execution-aware terms of the proxy and tighten turnover constraints, because the objective failed to encode feasibility.

Finally, cost inflation sensitivity should be summarized as a stability map: performance and risk metrics as functions of cost multipliers. A policy that remains acceptable across a reasonable band of multipliers is robust. A policy that only works at the baseline multiplier is fragile. This map becomes an institutional artifact: it can be reviewed, archived, and used as evidence in stage-gate decisions.

2.5.2 Sensitivity to tightened drawdown and turnover limits

Institutions do not operate with fixed constraints forever. Limits tighten after losses, during stress, or when capital allocation changes. A surrogate agent that only performs when limits are loose is not institutionally credible. Therefore, tightening limits is a core stress test of the proxy contract. The question is whether the policy's mechanism survives when governance becomes stricter, and whether the proxy objective and constraints interact coherently rather than producing pathological behavior.

Drawdown tightening is particularly important because drawdown is often the governance trigger that ends a strategy's autonomy. If a policy's behavior becomes unstable as the drawdown hard limit is reduced, this indicates that the policy's risk control is weak and that it relied on "room" in the drawdown budget. Tightening can be implemented by lowering the hard stop-out threshold, lowering the soft drawdown target, and increasing the penalty slope as drawdown approaches the limit. A robust policy should respond by reducing exposure earlier and by avoiding behaviors that produce long drawdown durations.

Turnover tightening tests operational feasibility. Institutions often impose stricter turnover limits due to capacity, transaction cost realities, or operational constraints. If the policy collapses when turnover caps are reduced, it suggests that the policy's edge depends on frequent trading. This can be acceptable in some contexts, but only if the operational model supports it. In many institutional mandates, excessive churn is disqualifying. Therefore, the proxy contract should reward mechanisms that do not require high-frequency turnover to survive.

These tightening tests also stress the proxy's internal consistency. A poorly designed proxy can create perverse incentives under tightened constraints. For example, if turnover is capped tightly but the objective still rewards rapid adaptation, the policy may attempt to "game" the cap by making larger position changes less often, increasing concentration risk. Or, if drawdown penalties are increased without proper scaling, the policy may become overly conservative, effectively going flat, which may be acceptable as a safety response but should be recognized explicitly.

The diagnostic outputs should therefore include: breach rates (which should remain near zero), time near constraints (which should not increase to extreme levels), exposure trajectories under tightening, and the decomposition of returns into gross, costs, and residual. If tightening reveals that the policy becomes dominated by constraints and loses its mechanism, this is not necessarily a failure; it may indicate that the strategy is not suitable under the stricter governance regime. The institutional decision then is "do not advance under these limits," which is valuable information.

Most importantly, tightened-limit sensitivity is a proxy stress test because it reveals whether the proxy objective leads to policies that are stable under governance. If small tightening causes collapse, the proxy may be too permissive or too reliant on soft penalties. This motivates Chapter 3's transition to constraint-respecting learning and explicit primal–dual governance signals.

2.5.3 Sensitivity to volatility regime prevalence and switching rates

A defining feature of markets is that regimes change, and the frequency and persistence of regimes are uncertain. A synthetic environment encodes a particular regime transition structure, but that structure is not truth; it is a modeling choice. Therefore, we must stress test the proxy and the resulting policies under variations in regime prevalence and switching rates. The question is whether the policy's behavior remains controlled when the world is more chaotic, more persistent, or simply different from the baseline.

Regime prevalence sensitivity changes how often the agent experiences high-volatility states, low-liquidity states, or crisis correlation states. A policy that is profitable only because stress regimes are rare is fragile. In the real world, stress can cluster. Therefore, the environment should be perturbed to increase stress prevalence and to cluster stress episodes. The policy should be evaluated for whether it de-risks appropriately, whether costs explode, and whether drawdowns remain bounded. If a policy collapses under higher stress prevalence, the proxy objective may not have adequately penalized stress fragility.

Switching-rate sensitivity changes the temporal structure of regimes. Faster switching creates choppier environments. In such environments, trend signals become noisy, and reactive strategies churn. Slower switching creates longer persistent regimes, which can favor certain mechanisms but can also create prolonged drawdowns if the policy is on the wrong side. Testing both extremes is important because both occur in real markets: calm periods can have long persistence, while transitions can be rapid and violent.

These tests also reveal whether the belief-state machinery is sound. If regime switching is faster than the belief filter's memory scale, the policy may be constantly behind, producing late reactions and churn. If switching is slower, an overly reactive filter may misclassify noise as transitions. Therefore, regime sensitivity tests should be paired with filter sensitivity tests. This produces an institutional insight: robustness depends not only on policy class but on the coupling between belief updates and control cadence.

From a proxy perspective, these tests ask whether the objective internalizes regime uncertainty. If the proxy rewards behavior that assumes stable regimes, it will produce policies that fail under switching. A governance-first proxy should encourage conservative behavior when regime uncertainty is high: reduce leverage, reduce turnover, and rely on robust signals rather than brittle conditional rules. If the policy does not do that, the proxy contract is incomplete.

The outputs should include regime-conditional metrics: performance and risk measured separately by regime, plus transition-period behavior. Transition periods often dominate risk because they are when signals reverse, costs rise, and beliefs update. If the policy fails primarily during transitions, this indicates a control instability. Such findings directly inform the design of Chapter 3's institutional overlays and constraint systems.

2.5.4 Sensitivity to liquidity collapse and capacity cliffs

Liquidity is the practical boundary of trading. Many strategies "work" only because the backtest assumes more liquidity than is available. Even sophisticated cost models can understate the nonlinear nature of impact. Therefore, a key stress test of the proxy objective is whether it produces policies that remain acceptable under liquidity collapse and capacity cliffs.

Liquidity collapse can be simulated by reducing market depth, increasing spreads, increasing impact coefficients, and steepening the impact exponent. The stress should be state-dependent: liquidity collapses more in high volatility and in crisis regimes. The policy should be evaluated for whether it responds by trading less and holding smaller exposures. A policy that continues to churn under liquidity collapse is not feasible and indicates that the proxy did not encode execution realism strongly enough.

Capacity cliffs are the nonlinear phenomenon where marginal costs rise rapidly beyond a certain participation level. A proxy that uses linear impact can be exploited by high-turnover policies. Under a convex impact model, these policies collapse. Therefore, a capacity-cliff stress test should explicitly steepen convexity and tighten participation limits. The goal is to observe whether the policy's performance degrades smoothly or collapses abruptly. Abrupt collapse indicates that the policy operated near a cliff and relied on optimistic convexity assumptions. This is a proxy exploit surface.

Liquidity collapse tests also interact with governance constraints. In stress, a prudent institution tightens turnover and exposure limits because feasibility shrinks. Therefore, the stress test should not merely inflate costs; it should tighten effective limits as well. This replicates real institutional behavior: risk budgets shrink when liquidity deteriorates. A policy that only survives under fixed limits is not institutionally aligned.

The diagnostic artifacts should include feasibility metrics: fraction of time in "tradable region," average participation, and instances where the policy attempted to trade but was constrained. These metrics are critical because they separate "the policy wants to do X" from "the policy can do X." A policy that repeatedly attempts infeasible trades is not only untradeable; it is also a governance risk because it indicates the policy's internal incentives are misaligned with feasibility.

If liquidity collapse reveals that the proxy is too permissive, remediation options include increasing cost conservatism, adding explicit penalties for participation and turnover, adding state-dependent scaling of risk, and strengthening hard constraints. This is one of the most direct pathways from

Chapter 2 evidence to Chapter 3 design upgrades.

2.5.5 Identifying "proxy-critical" parameters

A mature outcome of stress testing is the identification of proxy-critical parameters: parameters whose misspecification changes the ranking of policies or changes the acceptability conclusion. Proxy-critical parameters define the fragility surface of the proxy contract. They answer a practical institutional question: "which assumptions must be correct for this agent to be acceptable?" If acceptability depends on assumptions that are difficult to validate, then the agent is not ready for deployment.

Proxy-critical parameters often include execution parameters (spread multipliers, impact exponent), tail event intensity (jump probability, crisis severity), regime transition rates, and correlation stress levels. They can also include governance parameters such as drawdown limits and turnover caps. The goal is to map how acceptance changes as these parameters move. This mapping is not purely academic. It becomes a governance artifact that informs monitoring and escalation: if a parameter drifts in reality (for example, spreads widen beyond assumed levels), the map predicts that the policy's feasibility may deteriorate, triggering de-risking.

Identifying proxy-critical parameters also helps focus validation effort. Institutions cannot validate everything. They must prioritize. If the policy is highly sensitive to cost convexity, then validating cost models and monitoring liquidity becomes central. If the policy is highly sensitive to regime switching frequency, then monitoring regime indicators becomes central. If the policy is highly sensitive to crisis correlation, then correlation stress monitoring becomes central. The stress tests therefore translate into operational monitoring requirements.

The methodology for identifying proxy-critical parameters is systematic sensitivity analysis. One varies one parameter at a time and then varies combinations to detect interactions. One records not only performance but constraint breaches, tail losses, and boundary time. One then identifies regions where the policy remains acceptable and regions where it fails. The result is a feasibility map for the proxy contract.

Crucially, this exercise reveals whether the proxy objective is too "knife-edge." If small parameter changes produce large acceptability changes, the proxy is brittle. Brittle proxies invite reward hacking because the optimizer can find policies that exploit the knife-edge. Robust proxies produce policies whose behavior changes smoothly with parameters and whose acceptability region is broad.

In Chapter 2, proxy-critical parameter identification is not the end goal, but it is a major deliverable. It turns vague concerns about robustness into specific, testable claims and operational monitoring requirements. It also sets the stage for Chapter 3, where constrained optimization and primal–dual methods can be used to make acceptance less sensitive to uncertain parameters by enforcing constraints directly rather than relying on finely tuned penalty weights.

2.5.6 Synthesis: stress testing as proxy validation, not policy punishment

Stress testing the proxy itself reframes the purpose of robustness. The goal is not to punish policies for failing in extreme scenarios. The goal is to validate the proxy contract as an incentive system. If a policy fails under stress, the question is whether the failure reveals a loophole in the proxy or an unavoidable trade-off in the strategy mechanism. If it is a loophole, the proxy must be strengthened. If it is an unavoidable trade-off, the institution must decide whether the mechanism is acceptable under its mandate.

This is why Chapter 2 insists on stress testing before advancing. Without stress testing, optimization pressure will produce impressive but fragile policies. With stress testing, the development process becomes an institutional learning loop: proxies are revised, constraints are strengthened, and policies become more controlled. The stress suite produces evidence that supports stage-gate decisions. This is what makes surrogate agent development institutional rather than speculative.

2.6 Diagnostics and Exploitation Detection

2.6.1 Breach counts, near-breaches, and boundary time

Institutional governance begins with constraints. But evaluating constraint compliance is more subtle than counting breaches. A policy that breaches a hard limit even once may be disqualifying, but a policy that never breaches can still be unacceptable if it lives on the edge. Boundary behavior is where the system is most fragile because measurement error, slippage, or unmodeled stress can turn “almost compliant” into “non-compliant.” Therefore, a proper diagnostic battery includes three related quantities: breach counts, near-breaches, and boundary time.

Breach counts are straightforward: how often did the policy violate a hard limit such as maximum leverage, maximum turnover, or maximum drawdown. However, breaches should be categorized by severity and by cause. Was the breach caused by a sudden gap move (state shock) or by persistent risky positioning (policy choice)? Did the breach occur during a specific regime transition? Did it occur due to execution mismatch? These distinctions matter because they change remediation: one might tighten stop-out logic for gap moves, or one might redesign the objective to discourage persistent boundary loading.

Near-breaches capture risk that breach counts miss. A near-breach is a state where the policy is within a defined buffer of a limit, such as within 5% of leverage cap or within a small margin of drawdown limit. The buffer should be chosen institutionally: it represents the uncertainty margin that supervision teams require. Near-breaches matter because they reveal boundary seeking even when no violations occur. Under optimization pressure, policies often learn to operate just inside limits to maximize reward. This can look “efficient” but it is fragile. Near-breach rates quantify this behavior.

Boundary time aggregates the near-breach concept into a temporal measure: what fraction of time does the policy spend near limits? This metric is extremely valuable because it distinguishes occasional boundary contact from persistent boundary living. Institutions often tolerate occasional spikes near limits if they are rare and explainable, but they do not tolerate policies that effectively run at maximum risk all the time. Boundary time therefore becomes a governance signal that is often more important than average leverage or average turnover.

These diagnostics are also exploitation detectors. If a policy improves performance by increasing boundary time, it may be exploiting slack in the proxy objective. If tightening limits slightly causes performance collapse, it indicates that the policy's mechanism depends on boundary living. Such a policy is not robust. Therefore, Chapter 2 treats boundary metrics as first-class outputs, logged and compared across baselines and candidate policies.

2.6.2 Shadow prices and binding constraints as governance signals

In constrained optimization and in penalty-based objectives, constraints have dual meanings. In a constrained formulation, constraints can be associated with Lagrange multipliers that quantify how costly it is to satisfy the constraint. In a penalty formulation, constraint penalties function similarly: they indicate how much the policy is being "pushed back" from violating the constraint. These quantities—often called shadow prices in operations research—are powerful diagnostics because they reveal which constraints matter and how the policy's incentives interact with them.

A binding constraint is one that is active: the policy would violate it if the constraint were relaxed. Binding constraints reveal where the policy is pushing against governance. In an institutional context, this can be acceptable if the constraint is a budget constraint (e.g., a soft risk target) and if behavior remains controlled. But it is a warning if the constraint is a hard limit and the policy is constantly pressing against it. Binding constraints also reveal proxy misalignment: if a constraint is always binding, it may mean the objective encourages behavior that is incompatible with the constraint. The result can be unstable: the policy oscillates between pushing forward and being pushed back.

Shadow prices quantify sensitivity. If the shadow price of turnover constraint is high, it means the policy's objective value would increase significantly if turnover were allowed to rise. This suggests that the policy is turnover-hungry and may be exploiting weak cost modeling. If the shadow price of drawdown constraint is high, it suggests the policy would take more risk if allowed, indicating boundary seeking on risk. Shadow prices therefore provide an interpretable signal of what the optimizer "wants."

For institutional auditability, shadow prices are valuable because they can be tracked across runs. If a policy revision reduces shadow prices under stress, it indicates improved alignment: the policy is not constantly trying to violate constraints. Conversely, if shadow prices increase as performance improves, it suggests performance gains were bought by pushing harder against governance. This is exactly the kind of trade-off an institution must see explicitly.

In Chapter 2, shadow prices and binding constraints are used as exploitation detectors. They reveal whether optimization is producing policies that survive only because constraints prevent them from doing something worse. A policy whose performance depends on the constraint acting as a brake is not necessarily safe; it may be a “caged” exploit. The institution then asks: what happens if the brake is slightly weaker due to operational delays or model error? If failure is likely, the system is fragile. This motivates redesign of the proxy and, in Chapter 3, constraint-respecting learning that internalizes constraints rather than fighting them.

2.6.3 Cost-dominance flags and churn indicators

Execution is where proxy exploitation hides most easily, and it is also where diagnostics can be most revealing. A cost-dominance flag is a governance indicator that costs are consuming an unacceptable share of gross returns or that strategy behavior is driven by trading rather than by genuine edge. The purpose of cost-dominance analysis is not to “blame costs,” but to detect whether the policy’s mechanism depends on frictionless assumptions.

A first diagnostic is the *cost-to-gross ratio*: total execution costs divided by gross P&L. If this ratio is high, the strategy is paying heavily for its returns. Under modest cost inflation, net performance may vanish. A robust policy should maintain a healthy margin between gross returns and costs, especially under stress multipliers. A second diagnostic is the *turnover elasticity*: how much turnover changes when costs are inflated. A sensible controller should reduce turnover when costs rise. If turnover does not decrease, the policy is insensitive to execution, indicating a proxy weakness.

Churn indicators include the distribution of trade sizes and trade frequency. One should examine not only mean turnover but tail turnover: the 95th percentile of daily turnover, the maximum turnover event, and the clustering of high-turnover episodes. High-turnover clustering is particularly important because it often coincides with regime transitions and stress, when costs are highest. A policy that churns exactly when liquidity is worst is fragile and indicates misalignment between control and feasibility.

Cost-dominance also relates to boundary time. A policy can “hide” churn by staying under a turnover cap but still trading aggressively up to the cap. Therefore, one should measure turnover boundary time: fraction of time turnover is close to cap. If this boundary time increases with optimization, it suggests the optimizer is exploiting turnover slack.

Institutional acceptance requires that cost-dominance and churn be bounded and explainable. If performance comes from micro-trading effects under optimistic costs, the strategy is not production-grade. Chapter 2 uses these diagnostics to classify policy behavior. If a policy is churn-driven, the remediation is to strengthen cost realism, tighten turnover caps, and possibly change the policy class toward smoother controllers. If a policy remains profitable with low churn and controlled cost ratios, it is evidence of a more robust mechanism.

2.6.4 Tail concentration flags: CVaR and extreme-loss clustering

Tail risk is the most important hidden failure mode of optimized proxies because tail events are rare and can be missed in finite samples. Therefore, diagnostics must detect not only the magnitude of tail losses but their structure. Tail concentration flags are indicators that the policy's losses cluster in certain states, such as crisis regimes, correlation spikes, or liquidity collapses. Such clustering suggests the policy is carrying a hidden exposure that only manifests in those states.

Conditional Value-at-Risk (CVaR) is a central tail diagnostic because it measures expected loss in the worst fraction of outcomes. However, CVaR must be computed on appropriate distributions. It should be computed on returns net of costs, and it should be computed across scenario packs that include tail events. CVaR computed on a calm sample can be misleading. Therefore, Chapter 2 insists on CVaR in stressed distributions.

Extreme-loss clustering adds another dimension. Instead of treating tail losses as independent, we measure whether extreme losses cluster in time. Clustering matters because clustered losses can trigger drawdown gates and create cascades. A strategy that experiences one isolated large loss may recover; a strategy that experiences a sequence of large losses is more likely to be stopped out and to create operational crises. Therefore, diagnostics should include metrics such as maximum drawdown duration, number of extreme-loss days in a rolling window, and the correlation between extreme-loss days and turnover spikes.

Tail concentration flags also include *skewness* and *kurtosis* of returns, but with a governance framing. Negative skew can indicate tail selling. High kurtosis can indicate rare extreme events. These are not disqualifying per se, but they must be understood and bounded. If optimization increases negative skew while improving Sharpe, that is a classic sign of proxy exploitation: the policy improved smoothness by selling tail risk. Chapter 2 uses this pattern explicitly as a warning signal.

An institution may also require stress-specific tail metrics: CVaR conditional on crisis regimes, maximum loss conditional on liquidity collapse scenarios, and worst-case P&L across adversarial perturbations. These metrics directly tie tail risk to the states where it matters. A policy that fails in crisis regimes is not robust, regardless of its average performance.

2.6.5 Generalization gaps: train/test divergence under stress

A final diagnostic pillar is generalization: does the policy's behavior and performance persist outside the environment draw or seed used in optimization? Generalization gaps are the difference between training performance and testing performance, but in this context training and testing are not static datasets. They are different synthetic realizations, different regime sequences, and different stress configurations. A policy that performs well only on the optimization seed is likely overfit or exploitative.

Generalization must be evaluated across multiple axes. First, across random seeds: different noise realizations, different tail event timings. Second, across regime structures: different switching rates or regime prevalences. Third, across execution conditions: different cost multipliers or liquidity paths. A robust policy should maintain acceptable performance and constraint compliance across these axes. If performance collapses on a subset, that indicates conditional fragility.

Generalization gaps are especially important for learned nonlinear policies, but they also matter for tuned hand-built controllers. Selection effects can overfit even simple policies. Therefore, institutional-grade governance requires that the optimization process include holdout conditions: some seeds and stress scenarios are reserved for final evaluation and not used in tuning. This reduces the risk of p-hacking the stress suite itself.

Diagnostics should report not only mean gaps but distributions: how often does test performance fall below baseline? How often does the policy violate constraints in test but not in training? What is the worst-case test outcome? Institutions care about worst cases. A policy that is good on average but occasionally fails catastrophically is not acceptable.

In a surrogate-agent program, generalization diagnostics also inform proxy revision. If many policies overfit, it may indicate the proxy objective is too easy to hack, or the environment is too narrow. Broadening the environment, increasing perturbation diversity, and tightening acceptance criteria are then appropriate. This is another way in which diagnostics serve not only evaluation but system design.

2.6.6 Synthesis: diagnostics as institutional truth-telling

Diagnostics and exploitation detection are the truth-telling layer of the pipeline. Optimization will produce high scores. Diagnostics determine whether high scores mean robust mechanism or proxy exploitation. Breach and boundary metrics reveal governance pressure. Shadow prices reveal incentive conflicts. Cost-dominance flags reveal execution hacks. Tail concentration flags reveal hidden convexity exposure. Generalization gaps reveal overfitting and brittleness.

The institutional lesson is that metrics must be interpreted as a profile, not as a single number. A policy is acceptable not because it has a high return metric, but because its diagnostic profile shows controlled, feasible, explainable behavior under stress and across seeds. Chapter 2 codifies this profile into a diagnostic battery and requires it as an artifact bundle. This is what enables stage-gate decisions that are evidence-based rather than narrative-based, preparing the transition to Chapter 3's constrained learning framework.

2.7 Baseline Comparators and Null Models

2.7.1 Why strong baselines are mandatory under optimization pressure

Under optimization pressure, weak baselines are worse than no baselines. They create the illusion of progress by making it easy for an optimizer to "beat something," even if the resulting policy is fragile, exploitative, or operationally infeasible. In Chapter 2, we are explicitly studying Goodhart dynamics: what happens when a proxy objective becomes a target. In that setting, a baseline is not merely a benchmark. It is a governance instrument. It defines what "acceptable competence" looks like under the same constraints and the same execution assumptions, and it anchors interpretation when optimized policies produce impressive numbers.

A strong baseline must satisfy two requirements. First, it must be *credible*: it should represent a controller that a prudent institution would actually consider deployable in some form. Second, it must be *comparable*: it must operate under the same information set, the same feasibility constraints, and the same cost model as the candidate policies. If a baseline is unrealistically handicapped, the comparison is meaningless. If a baseline is allowed to cheat (for example, by using regime labels), it is also meaningless. The baseline is part of the laboratory contract; it must be governed just like the candidate.

Strong baselines are especially important because optimization can exploit measurement noise. A learned policy may appear better than a naive baseline simply because it overfits the particular environment draw. A strong baseline reduces the chance that "better" is merely "more overfit." Additionally, a strong baseline provides a diagnostic function: if a candidate policy beats the baseline only in calm conditions but loses under stress, the baseline reveals that the candidate's advantage is fragile. If a candidate beats the baseline primarily by increasing turnover or boundary time, the baseline exposes that the advantage was purchased by pushing harder against feasibility. These interpretations are impossible if the baseline itself is weak.

Finally, baselines matter for institutional communication. Committees do not want to hear that a strategy is "good because it is better than random." They want to hear that it is better than a conservative, supervised controller and that the improvement persists under stress and under tightened limits. Baselines provide the language of credibility: they show that the system is not only clever, but meaningfully better than disciplined alternatives.

2.7.2 Conservative controllers as anti-exploitation references

The first baseline class should be conservative hand-built controllers. Their role is to represent "what a cautious institution would do" when faced with the same partial observability and the same execution frictions. Conservative controllers are anti-exploitation references because they are less likely to discover proxy loopholes; they behave according to explicit rules that embed governance and feasibility. If a learned policy cannot beat such controllers under robust evaluation, it has not earned its complexity.

A canonical conservative baseline in surrogate trading laboratories is a volatility-targeting controller with bounded turnover and drawdown overlays. The mechanism is deliberately simple: scale exposure inversely with realized volatility to maintain a stable risk profile, limit position changes to avoid churn, and reduce exposure aggressively when drawdown approaches a threshold. This controller is not designed to maximize the proxy objective; it is designed to survive. It embodies institutional priorities: controlled risk, controlled turnover, and graceful de-risking in stress.

Another conservative controller can be a regime-aware overlay that uses belief states. For example, if the belief of crisis regime rises above a threshold, the controller scales exposure down and tightens turnover. This is not "learning"; it is governance encoded as policy. The purpose is not to outperform in calm conditions. The purpose is to demonstrate that a policy can remain controlled when uncertainty rises. Learned policies should be compared against this behavior, because institutions often require such overlays in production regardless of the alpha model.

Conservative baselines also help identify proxy exploitation. Suppose an optimized policy outperforms a conservative baseline but only by running at the leverage cap and trading aggressively. In that case, the baseline exposes the nature of the gain: the policy is purchasing return by consuming governance slack. The institution can then decide whether such behavior is acceptable. Often it is not. Therefore, conservative controllers are not only benchmarks; they are moral compasses for supervised behavior.

2.7.3 Robust MPC comparators and scenario-aware scoring

A second baseline class should represent "robust decision-making" rather than static rules: an MPC-style comparator. Model Predictive Control (MPC) is useful here not because it must be used in production, but because it embodies a disciplined alternative to black-box optimization. In MPC, the controller evaluates candidate actions by simulating short-horizon consequences under a set of scenarios, then chooses the action with the best robust score. This directly aligns with institutional supervision: decisions are justified by scenario analysis, and constraints can be enforced explicitly.

In a surrogate-agent laboratory, an MPC comparator can be implemented with a small discrete action set (e.g., a grid of exposure levels) and a scenario pack generated from the environment model and perturbations. For each candidate action, the controller computes a robust utility: expected return minus conservative cost estimates minus a tail penalty, possibly using worst-case or CVaR across scenarios. It then selects the action that maximizes this robust score subject to constraints. This is a strong baseline because it explicitly internalizes uncertainty and feasibility.

The governance value of MPC comparators is that they often resist proxy hacking. Because they evaluate across scenarios, they are less likely to choose actions that look good only in a narrow set of conditions. They also provide interpretability: one can inspect which scenarios dominated the decision and which constraints were binding. If a learned policy claims superiority, it must demonstrate superiority not only over simplistic rules but over a robust scenario-aware controller. This is a high bar, and that is precisely the point.

MPC comparators also provide a bridge to Chapter 3. Chapter 3 will introduce constraint-respecting learning and primal–dual methods. MPC demonstrates the institutional logic of constraint and scenario awareness. It shows that "smart" does not mean "opaque." A policy can be intelligent and still be justified in terms that supervision teams understand.

2.7.4 Ablations: removing one term from the proxy to reveal its role

Ablation baselines are often overlooked, but they are among the most powerful tools for diagnosing proxy alignment. An ablation is a controlled modification of the proxy objective or constraint system in which one term is removed or weakened. The resulting policy is then optimized under the modified contract and compared to the original. The purpose is to reveal what each term in the proxy actually does under optimization pressure.

For example, remove the tail-risk penalty term. If performance increases dramatically and tail risk explodes, this is strong evidence that the tail penalty was essential to prevent tail selling. Remove the execution convexity term or reduce cost multipliers. If performance increases and turnover explodes, this indicates that cost realism was essential to prevent churn exploitation. Remove the drawdown overlay. If the policy becomes unstable and experiences cascades, this indicates that drawdown controls were not cosmetic but structural.

Ablations also reveal interactions. Sometimes a term seems unimportant in isolation because another term partially compensates. For instance, a turnover cap may substitute for a convex cost term, or a drawdown penalty may substitute for a tail penalty in certain environments. Ablations can detect such substitutability and guide simplification: if two terms are redundant, governance may prefer the simpler contract. Conversely, if removing a term causes subtle but dangerous changes (e.g., more time near limits), then the term is critical even if headline metrics do not change much.

Institutional governance benefits from ablations because they translate abstract proxy design into concrete behavioral evidence. A committee can be shown: "without this penalty, the optimizer sells tails," or "without this constraint, the optimizer churns." This makes the proxy contract defensible. It also reduces the risk of accidental degradation: if later revisions change weights, ablation evidence helps ensure critical safeguards remain effective.

2.7.5 Null policies: flat, capped, and random-walk controllers

Null models complete the baseline suite by establishing "no-skill" references. In finance, null models serve two purposes. First, they ensure that apparent performance is not an artifact of environment drift or measurement error. Second, they calibrate how difficult the environment is: if a null policy performs surprisingly well, the environment may be too easy or mis-specified. If a null policy performs catastrophically, the environment may be too harsh or unrealistic.

A flat policy is the simplest null: maintain a constant low exposure or a balanced allocation with minimal trading. A capped flat policy respects constraints and costs. It represents the “do almost nothing” posture. In many environments, this policy will have modest performance and low turnover. It is useful because it sets a baseline for how much reward is available without active control. If an optimized policy barely beats a flat policy after costs, it is not compelling.

A second null is a capped random policy: choose actions randomly from the allowed action set while respecting hard limits. This policy should not have meaningful edge, but it reveals the distribution of outcomes that can arise purely from randomness under the environment and cost model. If an optimized policy’s performance is within the distribution of random outcomes, that suggests overfitting or lack of mechanism. In addition, random policies help calibrate tail behavior: if random actions frequently breach constraints, then the constraints or environment may be configured in a way that makes feasibility too narrow. That itself is a governance finding.

A third null is a random-walk controller that slowly drifts exposure, representing “uninformed adaptation.” It is more realistic than pure randomness because it produces smoother action paths. It helps detect whether the environment rewards merely changing exposure regardless of signal. If a drift policy performs well, the environment may be embedding exploitable structure unrelated to the intended mechanism. That indicates that the lab itself must be revised before optimizing sophisticated policies.

Null policies must be evaluated with the same metrics and diagnostics as candidates. Their purpose is not to look good; it is to provide calibration and sanity checks. Institutional-grade labs always include such checks because they prevent self-deception: if a policy only beats nulls marginally, it is not worthy of escalation.

2.7.6 Baselines as part of the audit record and stage-gate logic

The baseline suite is not only methodological; it is part of the governance artifact bundle. Every run should export baseline results alongside candidate results, using identical evaluation pipelines, scenario packs, and metrics. This enables reviewers to assess not only absolute performance but relative performance. It also enables longitudinal comparison: if environment configurations change, baselines reveal whether changes affected the difficulty of the task.

Baselines also shape stage-gate criteria. A common institutional criterion is: a candidate policy must outperform conservative baselines on net performance while remaining within the same risk envelope, and it must outperform robust comparators under stress. Additionally, it must not do so by increasing boundary time or by inflating tail exposure. Baselines therefore define what “advance” means. Without baselines, advancement becomes subjective and narrative-driven.

Finally, baselines are essential for remediation. If a candidate fails under stress, one can compare its failure mode to baseline behavior. If the candidate fails while baselines remain controlled, it indicates that the candidate's optimization introduced fragility. If baselines also fail, it indicates that the environment stress is extreme or that the proxy contract must be revised more fundamentally. Thus, baselines help separate "policy failure" from "lab failure," which is an important distinction in governed development.

In sum, Chapter 2 treats baselines and null models as institutional scaffolding. They prevent meaningless victories, expose proxy exploitation, calibrate environment realism, and provide the comparative evidence needed for supervisory decisions. They turn optimization results into interpretable outcomes and ensure that "better" means "better under governance," not merely "higher score under an imperfect proxy."

2.8 Training and Tuning Protocols Under Governance

2.8.1 Deterministic seeds and repeatability of "best" results

The first rule of institutional optimization is that "best" must be repeatable. If a result cannot be reproduced exactly, it cannot be audited, and if it cannot be audited, it cannot be advanced. This is especially important in surrogate-agent development because randomness is everywhere: synthetic market generation, scenario sampling, stochastic policies, randomized initialization, and numerical nondeterminism. Without careful controls, a "best policy" can be a lucky draw rather than a robust discovery.

Deterministic seeds provide the baseline control. Every component that draws randomness must be seeded: environment generation, policy initialization, training minibatch selection if any, scenario pack sampling, and perturbation generation. The run manifest must record these seeds and bind them to the configuration hash. Repeatability is then tested as a stage gate: rerun the exact configuration and confirm that metrics, logs, and exported policies match within strict tolerances. If they do not match, the pipeline is not governed enough to proceed.

However, institutional repeatability goes beyond seeds. The environment fingerprint must record versions of critical dependencies. Floating point and numerical libraries can introduce tiny differences that compound during training. Therefore, the governance process must treat deterministic replay as a capability that is continuously tested, not assumed. If perfect determinism is infeasible across hardware, then the institution must adopt tolerance-based determinism: the run is considered reproducible if outputs are within specified numerical bounds. The bounds themselves become part of governance, because they define what is considered "the same" result.

Repeatability also protects against accidental p-hacking. When teams rerun experiments and results change materially, it becomes easy to cherry-pick the best run. A governed process prevents this by requiring that a claimed result correspond to a specific run identity and that the run be replayable. This shifts the culture: instead of "we saw it once," the standard becomes "we can reproduce it and we can explain it."

2.8.2 Avoiding p-hacking: predefined search budgets and stopping rules

In finance, the most common form of reward hacking is not performed by the agent; it is performed by the research process. Trying many variants and selecting the best is an optimizer, and if the process is unconstrained, it will overfit. This is p-hacking by another name. Chapter 2 therefore treats the optimization budget as a governance object.

A search budget defines how many variants will be tried, what range of hyperparameters will be explored, and how decisions will be made. Budgets can be expressed as number of training runs, number of tuning trials, or total compute steps. The key is that the budget is defined *before* results are known, and it is logged as part of the run manifest. This ensures that the search is not expanded opportunistically until a desired outcome is found.

Stopping rules are equally important. Without stopping rules, optimization can continue indefinitely until a lucky result appears. In a governed pipeline, stopping rules define when optimization ends and evaluation begins. A stopping rule can be based on maximum trials, maximum compute, or convergence criteria. It can also include early stopping for instability: if breach rates exceed a threshold or if the policy exhibits pathological boundary time, the run is terminated and recorded as a failure. This prevents wasting compute on obviously exploitative behaviors and creates a disciplined record of what was attempted.

Avoiding p-hacking also requires separating development and evaluation. Some seeds and scenarios must be reserved as holdouts and not used for tuning decisions. Otherwise, the stress suite itself becomes a target. This is a subtle but critical point: once researchers know the stress tests, they can tune to pass them without being truly robust. Therefore, institutional pipelines often maintain a "secret" or at least "untouched" evaluation set. In the synthetic lab context, this can be implemented as reserved seeds and reserved perturbation configurations.

Finally, governance requires reporting the full search distribution, not only the winner. The deliverable should include summary statistics of the tried policies: median performance, variance, failure rates, and the rank of the selected policy. If the winner is an extreme outlier, it is more likely to be luck or exploitation. If many policies perform similarly, it suggests a robust region. This distributional reporting is one of the strongest defenses against p-hacking.

2.8.3 Hyperparameter governance: bounding and logging search spaces

Hyperparameters are assumptions. In trading, hyperparameters are often where hidden overfitting lives: thresholds, smoothing windows, penalty weights, learning rates, exploration temperatures. Under optimization pressure, these settings can be adjusted until performance appears excellent. Therefore, hyperparameter governance is a core requirement.

Bounding search spaces is the first control. Each hyperparameter must have a justified range. The justification can be conceptual (e.g., turnover cap must reflect operational feasibility) or empirical (e.g., volatility window must reflect realistic estimation horizons). The range must be encoded in a typed configuration graph and validated. This prevents nonsensical settings from being tried and prevents the search from drifting into extreme regimes that produce unrealistic behavior.

Logging search spaces is the second control. It is not enough to record the chosen hyperparameters; the pipeline must record which hyperparameters were eligible and which were actually tried. This is a provenance requirement. In institutional review, one must be able to say: "we searched within these boundaries, and we selected this setting." Without that, a strategy can be accused of cherry-picking, and the accusation may be correct.

A third control is hierarchical governance of hyperparameters. Some settings are "research" settings that can be tuned freely within bounds; others are "governance" settings that should not be tuned to improve performance. For example, drawdown hard limits and turnover caps often reflect institutional mandates and should be fixed. Allowing an optimizer to tune them undermines governance because the optimizer will relax constraints to improve reward. Therefore, the pipeline should separate policy hyperparameters (which can be optimized) from governance constraints (which are fixed) and treat any changes to governance constraints as requiring explicit human approval.

This separation is fundamental. It prevents the system from optimizing away supervision. It also ensures that improvements are meaningful: if performance improves while constraints remain fixed, the improvement is more credible than if performance improves because constraints were loosened.

2.8.4 Cross-seed validation: robustness across random environments

Cross-seed validation is the institutional response to noise and selection effects. Because the synthetic environment is stochastic, any single run is an incomplete story. Cross-seed validation evaluates policies across multiple independent environment draws, each defined by a seed. The goal is to measure not only mean performance but variability and worst-case outcomes. This is crucial because institutions care about stability and tail risk.

Cross-seed validation must be designed into the protocol. The number of seeds, the allocation of seeds to training versus evaluation, and the aggregation method must be predefined. A common pattern is: train on a small set of seeds, then evaluate on a larger set of held-out seeds. Another pattern is to train across multiple seeds simultaneously (domain randomization) to encourage robustness. Regardless of method, the protocol must be documented and reproducible.

Aggregating results across seeds is itself a governance decision. One can use mean metrics, but institutions often prefer conservative aggregates: lower quantiles of performance, upper quantiles of drawdown, worst-case CVaR. A policy that has high mean return but occasional catastrophic drawdowns may be unacceptable. Therefore, the acceptance criterion should reflect institutional risk tolerance: require that worst-case seed outcomes remain within limits, not only average outcomes.

Cross-seed validation also reveals exploitation patterns. A policy that exploits a narrow feature of one environment realization will not generalize. Its performance will vary wildly across seeds. High variance is therefore a warning. Conversely, a policy that reflects a stable mechanism will perform more consistently. This consistency is evidence of robustness.

In the audit bundle, cross-seed results should be exported as a table of per-seed metrics and diagnostics. This enables reviewers to see whether failures cluster in certain conditions and whether the policy is fragile. It also supports remediation: if failures occur primarily under certain regime sequences, the environment and proxy can be revised accordingly.

2.8.5 When to stop optimizing: institutional sufficiency thresholds

A final governance requirement is knowing when to stop. Unbounded optimization is not only a risk of p-hacking; it is a risk of increasing exploit sophistication. The more you optimize, the more likely you are to find degenerate solutions that exploit proxy loopholes. Therefore, institutional pipelines define sufficiency thresholds: conditions under which further optimization is not warranted.

Sufficiency thresholds can take several forms. One is performance sufficiency: if the policy meets the required return and risk profile relative to baselines, additional gains may not justify additional complexity. Another is robustness sufficiency: if the policy passes stress tests and cross-seed evaluations with stable behavior, the institution may prefer to stop rather than search for marginal improvements that could reduce robustness. A third is governance sufficiency: if the policy's diagnostic profile is clean (low boundary time, low tail concentration, low cost dominance), the institution may consider it acceptable and avoid chasing higher returns that could introduce hidden risk.

Sufficiency thresholds also include disqualifying conditions. If optimization consistently produces policies that rely on boundary living, tail selling, or churn, and if proxy revisions are not yet implemented, the correct decision may be to stop optimization and return to contract redesign. This is a key institutional discipline: do not continue optimizing a broken proxy. Fix the proxy first.

Finally, the decision to stop is a human governance decision, but it should be supported by artifacts. The pipeline should produce a stage-gate recommendation: advance, do not advance, or revise proxy. The recommendation should be based on predefined criteria, not on subjective impressions. This is how the optimization process remains supervised.

2.8.6 Synthesis: optimization as a governed experiment, not a search for bragging rights

Training and tuning protocols under governance reframe optimization as an experiment rather than a treasure hunt. The objective is not to find the maximum score at any cost. The objective is to apply controlled optimization pressure, observe how policies behave, detect proxy exploitation, and produce reproducible evidence that supports institutional decisions.

Deterministic seeds and replayability ensure that results are real and auditable. Search budgets and stopping rules prevent p-hacking and runaway exploitation. Hyperparameter governance ensures that supervision is not optimized away. Cross-seed validation ensures robustness is measured rather than assumed. Sufficiency thresholds ensure that optimization ends when evidence is sufficient, not when enthusiasm peaks.

These protocols are the bridge between Chapter 1's specification and Chapter 3's constrained learning. They ensure that the pipeline is disciplined enough to handle optimization without producing fragile, overfit, or exploitative agents. In institutional terms, they convert "we optimized an objective" into "we ran a governed optimization process with auditable evidence." That is the standard required for any surrogate agent to be considered for advancement.

2.9 From Optimization Evidence to Institutional Decisions

2.9.1 Advance criteria: what must be true to proceed to Chapter 3

At the end of Chapter 2, the central institutional question is not "did we find a high-scoring policy?" It is "did we produce sufficient evidence that optimization can be applied to this proxy contract without generating unacceptable behavior?" Chapter 3 escalates to constraint-respecting learning and institutional control logic. Proceeding requires that Chapter 2 establish a foundation of robustness, auditability, and incentive alignment. Therefore, advance criteria must be explicit and conservative.

A first advance criterion is *reproducibility*. The chosen candidate policies must be replayable under identical configurations with deterministic seeds and stable environment fingerprints. Metrics, logs, and exported policy parameters must match within defined tolerances. If the pipeline cannot reproduce "best" results, it cannot support Chapter 3's stronger claims because Chapter 3 will involve more complex optimization and therefore more potential sources of ambiguity.

A second advance criterion is *governance compliance*. Hard constraints must exhibit near-zero breaches, and near-breach boundary time must be within acceptable bounds. This criterion is deliberately stricter than “no breaches.” An institution does not want a policy that merely avoids breaches; it wants a policy that does not rely on razor-thin buffers. Therefore, advancement requires that leverage, turnover, drawdown, and any concentration limits are respected with conservative margins, including under scenario packs and cost inflation.

A third advance criterion is *robustness under stress*. Candidate policies must remain acceptable across the scenario pack: volatility inflation, liquidity collapse, correlation compression, jump events, and regime acceleration. “Acceptable” must be defined in terms of institutional thresholds: maximum drawdown below hard limit, CVaR below risk budget, cost-dominance below a defined ratio, and breach rates within tolerance. The policy does not need to “win” in all stresses, but it must fail gracefully: it must de-risk, reduce turnover, and remain within governance boundaries.

A fourth advance criterion is *generalization*. Performance and behavior must persist across held-out seeds and perturbed regime structures. A policy that is impressive on the training seed but fragile elsewhere is not a candidate. Institutions require evidence that the mechanism is stable and not a byproduct of one realization. Therefore, advancement requires that performance distribution across seeds is not dominated by a few outliers and that worst-case seeds remain within risk limits.

A fifth advance criterion is *comparative credibility*. Candidate policies must outperform strong baselines net of costs, with a diagnostic profile that is at least as clean. If a candidate outperforms by increasing churn, living at boundaries, or worsening tail exposure, it is not institutionally superior. The comparison must be made on the full profile: returns, risk, feasibility, and diagnostic signals. Only then is the candidate “worth optimizing further” in Chapter 3.

In summary, advancement requires that the proxy contract and the optimization harness produce controlled behavior under pressure. Chapter 3 is not a place to fix basic misalignment. It is a place to implement constrained learning on top of a contract that has already demonstrated defensibility.

2.9.2 Do-not-advance criteria: what failures are disqualifying

Equally important are disqualifying criteria: conditions under which the correct decision is to stop and redesign rather than to proceed. Institutions must treat “do not advance” as a valid and often valuable outcome. In surrogate-agent development, disqualifying failures often signal proxy exploitation or feasibility mismatch, and proceeding would only amplify those problems.

A first disqualifier is *systematic proxy exploitation*. If optimized policies repeatedly achieve high scores by selling tails, increasing churn, exploiting weak cost models, or leveraging unrealistic execution assumptions, then the proxy contract is not safe under optimization. One can often detect this by seeing that performance collapses under modest cost inflation, that skew becomes strongly negative, that CVaR worsens while Sharpe improves, or that turnover boundary time rises sharply. If these patterns persist across multiple optimization runs, the proxy is structurally exploitable. Proceeding to Chapter 3 would likely produce even more sophisticated exploitation.

A second disqualifier is *constraint fragility.* If the policy's compliance depends on delicate cancellations or on being just inside limits, then small operational errors would create breaches. High boundary time, high shadow prices, and frequent near-breaches are signals. Institutions often require buffers; therefore, a policy that lives at the boundary is disqualifying unless the mandate explicitly tolerates such behavior and the risk system can enforce real-time constraints with minimal latency.

A third disqualifier is *catastrophic tail behavior under stress.* If scenario packs produce drawdown cascades that violate limits or produce unacceptable CVaR, the policy is not robust. Even if such failures are rare, institutions are often unwilling to accept strategies that can blow up under plausible stress. The entire point of synthetic stress is to surface these risks early. If stress reveals unacceptable tails, the strategy mechanism may be disqualified or the proxy must be redesigned to penalize tail exposure more strongly.

A fourth disqualifier is *non-reproducibility and unstable optimization.* If optimization outcomes vary wildly across reruns, if best results cannot be replayed, or if training becomes unstable (oscillating objectives, exploding multipliers, inconsistent constraint enforcement), then the pipeline is not governed enough to proceed. Chapter 3 will increase complexity; instability at Chapter 2 is a warning that governance infrastructure is inadequate.

A fifth disqualifier is *lack of comparative advantage.* If optimized policies do not outperform robust baselines or do so only marginally and unreliably, then the complexity is not justified. Institutions have a strong preference for simplicity when performance differences are small. If a conservative baseline is nearly as good and has cleaner diagnostics, the institution may prefer it and stop the surrogate agent escalation path.

These disqualifiers are not failures of effort; they are successful detections of risk. They preserve institutional capital and credibility by preventing fragile systems from advancing.

2.9.3 Remediation loop: revising the proxy contract

When the evidence indicates that the proxy is exploitable or fragile, the correct institutional response is remediation: revise the proxy contract and rerun governed optimization. This remediation loop is the essence of surrogate agent development under governance. The proxy is a hypothesis. Optimization and stress testing are falsification attempts. Remediation is the process of strengthening the hypothesis.

Remediation begins by diagnosing the failure mode. If the policy sells tails, strengthen tail penalties, introduce CVaR constraints, increase tail scenario frequency, and ensure that tail events appear in evaluation. If the policy churns, strengthen convex cost modeling, add explicit turnover penalties, and enforce tighter turnover caps. If the policy overfits regimes, broaden regime variability, add structural breaks, increase observation noise, and restrict features to reduce leakage. If the policy concentrates under correlation stress, add correlation shock scenarios and enforce concentration constraints.

A key governance requirement is to avoid “weight fiddling.” Naively adjusting penalty weights can produce unstable objectives and can itself become a tuning knob that invites p-hacking. Therefore, remediation should prioritize structural changes: add hard constraints where needed, redesign cost models to be conservative by construction, and redesign evaluation to include stresses that cannot be ignored. Penalty weights can be adjusted, but they should be adjusted under explicit governance, with ranges justified and changes logged.

Remediation also includes simplifying where possible. Sometimes a proxy is exploitable because it is too complex and hides loopholes. A simpler objective with clear, conservative terms can be safer. The remediation loop therefore includes ablation evidence: remove terms, test behavior, and keep only what is essential for alignment.

Finally, remediation must be documented as part of the institutional record. The revision should have a rationale: “we observed churn exploitation under cost inflation; we revised the cost model and tightened turnover cap.” This documentation turns the development process into a governed narrative of improving alignment, which is critical for committee review.

2.9.4 Documentation: failure modes as formal outputs

Institutional decisions require documentation that is both precise and reviewable. Chapter 2 therefore treats failure documentation as a deliverable. If a policy fails, the failure must be recorded with run identity, configuration hash, environment fingerprint, and the full diagnostic profile. The documentation should state what failed, under which scenario, and why it matters.

A disciplined failure document includes: (i) a short description of the policy and optimization method; (ii) the baseline comparison; (iii) the stress scenario where failure occurred; (iv) the metrics that triggered disqualification (e.g., drawdown breach, CVaR breach, cost dominance); (v) diagnostics indicating exploitation (boundary time, shadow prices, turnover spikes); and (vi) a remediation hypothesis. This is not a blame document. It is a scientific artifact. It increases institutional learning.

Failure documentation also supports governance escalation pathways. In an institution, certain failures trigger mandatory review. For example, a drawdown breach may require risk committee attention. A tail-loss cluster may require model risk review. Chapter 2’s documentation can be structured to align with these pathways: each failure type maps to a governance category and a review owner. This is the foundation for production readiness, where monitoring will trigger similar escalations.

Finally, documenting failures prevents the most dangerous institutional behavior: forgetting. Teams often move on from failed experiments without preserving the evidence, which leads to repeating mistakes. A governed surrogate agent program treats failure evidence as a memory system. It prevents the organization from relearning the same Goodhart lessons repeatedly.

2.9.5 Committee-facing summary: evidence over narrative

The culmination of Chapter 2 is a committee-facing summary that supports an advance or do-not-advance decision. This summary must resist narrative temptation. It must be structured around evidence. The committee does not need every technical detail, but it needs clear answers to supervisory questions.

A committee-facing summary should include: (i) what proxy contract was optimized; (ii) what optimization budgets were used; (iii) how reproducibility was ensured; (iv) how baselines performed; (v) how candidate policies performed relative to baselines; (vi) stress test outcomes and any disqualifying failures; (vii) diagnostic profiles indicating whether gains came from robust mechanism or exploitation; and (viii) the stage-gate recommendation. The summary should also state the limitations explicitly: synthetic environment, not validated on real markets, human review required. Institutional credibility depends on acknowledging limits.

A key institutional practice is to separate "facts" from "interpretation." Facts are metrics, breaches, stress outcomes. Interpretations are hypotheses about mechanisms and remediation plans. The summary should present both, but clearly separated. This prevents misunderstandings and makes review more disciplined.

The final decision is then framed properly: advancing to Chapter 3 is not a celebration of high returns; it is a controlled escalation because the system demonstrated robustness under optimization pressure and because governance artifacts support supervision. Conversely, not advancing is not failure; it is an evidence-based decision that the proxy contract or environment must be revised before applying stronger optimization.

2.9.6 Synthesis: Chapter 2 as a stage-gate discipline

Chapter 2's institutional value is that it turns surrogate agent development into a stage-gated process. Optimization evidence is interpreted through robustness, diagnostics, and governance criteria. The outcome is a disciplined decision: proceed, revise, or stop.

This is what makes the three-chapter journey institutional-grade. Chapter 1 defines the contract and the lab. Chapter 2 stress-tests the contract under optimization pressure and produces a defensible stage-gate decision. Chapter 3 then implements constrained learning and institutional control logic on top of a validated contract. Without Chapter 2's decision discipline, the program would drift toward optimizing impressive but fragile proxies. With it, the program becomes a supervised pipeline for building surrogate trading agents that can be evaluated, audited, and governed.

2.10 Chapter 2 Notebook Deliverables

2.10.1 Deliverables: optimization harness and search budgets

The first deliverable of the Chapter 2 notebook is a complete optimization harness that is governed by design. This harness is not merely "code that trains an agent." It is an institutional process encoded in software: it defines what can be optimized, how much optimization is allowed, how randomness is controlled, what artifacts must be produced, and how results are selected and reported. In other words, it is the mechanism by which optimization pressure is applied in a way that remains auditable.

At minimum, the optimization harness exports a typed configuration object that includes: the proxy objective definition, the constraint system, the environment generator specification, and the search protocol (budgets, stopping rules, parameter ranges). The harness then implements one or more optimization routes—such as policy parameter tuning, robust MPC parameter tuning, or constrained policy learning—and binds each run to a deterministic seed set. The key requirement is that each run has a unique run identity with UTC timestamps, a configuration hash, and an environment fingerprint. These are written to a run manifest and stored alongside outputs.

Search budgets are central. The notebook must explicitly encode budgets such as: number of trials, maximum compute steps, maximum wall-clock proxy (if relevant), and maximum number of candidate policies retained. It must encode stopping rules and early termination criteria for pathological behavior. For example, if a candidate exceeds breach thresholds or exhibits extreme boundary time early, the run can be terminated to preserve compute and to create a recorded failure artifact.

The harness must also output the *search record*, not just the winner. This includes a summary of all trials: parameter settings, key metrics, failure flags, and diagnostic signals. This record is essential because it transforms the search from a story into evidence. It also allows reviewers to see whether the selected policy is representative or an outlier.

Institutionally, this deliverable is what makes the notebook production-ready: it encodes how optimization is governed, ensuring that future reruns follow the same discipline. Chapter 3 will build on this harness, but Chapter 2 must establish it.

2.10.2 Deliverables: robustness and adversarial scenario engine

The second deliverable is the robustness engine: the scenario pack generator and the adversarial perturbation mechanism. This engine operationalizes the definition of robustness. It must generate a structured suite of stress configurations that degrade volatility, liquidity, correlation structure, and jump behavior. It must also generate bounded adversarial perturbations that represent plausible model error in dynamics, observation noise, and execution costs.

This deliverable includes: (i) a scenario pack specification file (versioned), (ii) deterministic generation of scenario instances given seeds, (iii) a registry of scenario categories and parameter ranges, and (iv) the ability to run any policy through the full pack and produce standardized outputs. The engine must support both baseline scenarios (representing expected conditions) and stress scenarios (representing adverse conditions). It should also support "tightened governance scenarios," where effective limits are reduced, reflecting real institutional behavior under stress.

The adversarial engine must be auditable: perturbation budgets must be explicit, and the mechanism must record which perturbations were applied in each evaluation. This enables a critical institutional artifact: worst-case or near-worst-case performance summaries under bounded uncertainty. Even if the program does not implement full min-max optimization, the evaluation should report how sensitive outcomes are to adversarial perturbations.

The robustness engine is also a diagnostic engine. It should be able to attribute failure: which scenario category triggered breaches, which perturbation dimension caused collapse, and which metrics signaled exploitation. This attribution is part of the deliverable because it enables remediation: it tells the team what to fix in the proxy contract and what to monitor in production.

2.10.3 Deliverables: diagnostic battery and exploitation flags

The third deliverable is the diagnostic battery: a standardized set of metrics and flags designed to detect proxy exploitation and fragility. The battery must produce not only summary statistics but behavioral signatures. It should include breach counts, near-breaches, boundary time, turnover distributions, cost-to-gross ratios, tail metrics (CVaR, worst-case), drawdown clustering, and generalization gap measures across seeds and scenarios.

Exploitation flags are the key institutional layer. A flag is a rule that marks behavior as suspicious or disqualifying. Examples include: (i) cost dominance above a threshold, (ii) turnover boundary time above a threshold, (iii) negative skew beyond a threshold combined with high Sharpe (tail-selling signature), (iv) CVaR exceeding budget, (v) repeated near-breaches in drawdown or leverage, and (vi) large train/test divergence. The thresholds should be explicit and stored in the configuration.

The diagnostic deliverable should be exported as structured JSON artifacts. Each run's diagnostics must be stored under a run$_{i}ddirectory, includingper-scenarioandper-seedbreakdowns.Thisiscrucialforauditabi$
$reviewerscantraceexactlyhowtheconclusionswerederived.Thenotebookshouldalsooutputahuman-$
$readablereportsummarythatreferencestheseartifacts, buttheauthoritativerecordshouldbemachine-$
$readable.$

This deliverable is what prevents the pipeline from being driven by headline returns. It ensures that the institution can evaluate policies by their full behavioral profile and can reject policies that score well by hacking the proxy.

2.10.4 Deliverables: comparative baselines and ablation suite

The fourth deliverable is the comparative baseline suite and ablation suite. The notebook must implement and evaluate: conservative controllers, robust MPC comparators, and null policies. It must run these baselines through the same robustness engine and diagnostic battery as candidate policies, producing comparable artifact bundles. Baselines must be first-class citizens in the evaluation pipeline.

The ablation suite is equally important. It must enable controlled modifications of the proxy contract: remove or weaken one term at a time and rerun optimization under the modified contract. The ablation results reveal which proxy terms are critical and which are redundant. They also reveal where exploitation pressure concentrates when safeguards are removed. This is essential institutional evidence because it makes the proxy contract defensible: the institution can demonstrate that specific terms prevent specific exploit behaviors.

The deliverable includes a comparative table of baseline and candidate results across metrics and diagnostics, as well as distributional plots or summaries across seeds (in a no-pandas environment, this can be exported as arrays and summarized). The key is that comparisons are standardized and reproducible. A committee should be able to see: candidate vs baseline under baseline conditions and under stress, with diagnostics.

The comparative suite also informs stage-gate decisions. If candidate policies do not outperform baselines robustly, the institution can stop. If candidate policies outperform but with worse diagnostics, the institution can revise the proxy. If candidate policies outperform with clean diagnostics, the institution can proceed. The baseline suite makes these decisions evidence-based.

2.10.5 Deliverables: audit bundle and stage-gate recommendation

The fifth deliverable is the audit bundle: the complete set of artifacts needed to review, reproduce, and justify the Chapter 2 outcome. This bundle includes: run$_manifest.json(run_id, UTCtimestamps, confighash, envf$
$trial - by - trialsummary.$

Critically, the notebook must output a stage-gate recommendation object. This object is not a narrative; it is a structured decision artifact that states one of: ADVANCE, DO_NOT_ADVANCE, or REVISE_PROXY. It should include justification fields: which criteria were met, which were not, and which failures (if any) were disqualifying. It should also include remediation suggestions when appropriate, tied to observed exploitation patterns.

The stage-gate object is the institutional bridge between Chapter 2 and Chapter 3. It allows the project to proceed only when evidence is sufficient. It also creates a permanent record of why a decision was made, which is essential for governance.

2.10.6 Synthesis: what the Chapter 2 notebook produces in institutional terms

In institutional terms, the Chapter 2 notebook produces a governed optimization and evaluation pipeline that can withstand scrutiny. It produces: (i) a controlled way to apply optimization pressure, (ii) a robust stress environment that tests proxy fragility, (iii) diagnostics that detect exploitation, (iv) baselines and ablations that anchor interpretation, and (v) audit bundles and stage-gate recommendations that support supervision.

The purpose is not to deliver "the best trading agent." The purpose is to deliver the evidence and infrastructure needed to decide whether any optimized surrogate agent is credible enough to justify further escalation. That is the institutional definition of production readiness at this stage. Chapter 3 will deepen the control and constraint logic, but Chapter 2 must already function as a supervised gate: it must detect Goodhart failures, document them, and prevent fragile systems from advancing.

2.11 Transition to Chapter 3: Constrained Control as the Institutional Upgrade

2.11.1 Why robustness testing is not enough without constrained learning

Chapter 2 established the disciplined posture that an institution must adopt when optimizing proxy objectives: apply optimization pressure, stress the resulting behavior, and reject policies that exploit measurement blind spots. Robustness testing is essential, but by itself it is not sufficient. The reason is structural: robustness testing evaluates outcomes after the fact, while optimization happens before the fact. If the training or tuning process is allowed to optimize a proxy with only soft penalties, it can repeatedly generate policies that are "barely acceptable" in the baseline environment and then fail when stressed. Robustness testing will catch those failures, but the development process becomes a cycle of discovering fragility rather than preventing it.

In practical terms, robustness testing can tell you that a policy sells tails, churns under weak costs, or lives at the boundary. But if the optimizer is still free to trade these behaviors against reward using penalty weights, the optimizer will keep finding them, especially as it becomes more capable. Penalty-based objectives create an implicit exchange rate between performance and violations: "how much reward am I allowed to buy by tolerating more turnover, more drawdown risk, or more tail exposure?" Institutions often reject that exchange rate. Certain constraints are not negotiable. They are not "preferences"; they are mandates. That is why the institutional upgrade must move from evaluation-only robustness toward constraint-respecting learning: the optimization process itself must be shaped so that constraint compliance is part of the solution, not a post-hoc filter.

2.11.2 From penalties to constraints: CMDP framing

The conceptual shift that enables this upgrade is to frame the surrogate agent problem as a constrained control problem rather than a pure reward maximization problem. In reinforcement learning terms, this is the Constrained Markov Decision Process (CMDP) framing. Instead of specifying only a reward function to maximize, we specify reward *and* a set of cost signals (or constraint signals) that must remain below thresholds. In trading, these constraint signals naturally correspond to institutional risk and feasibility limits: drawdown budgets, turnover caps, leverage caps, concentration limits, and tail-risk budgets.

The CMDP framing matters because it changes what the optimizer is allowed to do. Under penalties, the optimizer can decide that violating a constraint is worth it if the penalty is not large enough. Under constraints, violation is not part of the feasible plan. The optimizer must find policies that satisfy the constraints (or satisfy them with bounded, explicitly controlled violations in soft-constraint settings). This aligns directly with institutional supervision: the institution defines hard limits, and learning occurs inside those limits.

CMDPs also clarify accountability. The proxy objective becomes a contract with two parts: a performance objective and a constraint system. The constraint system is not tuned to improve performance; it is specified by governance. This separation prevents "optimizing away supervision" and makes the learning process defensible. Chapter 3 will operationalize this separation by implementing training that monitors constraint costs, enforces limits, and produces artifacts that demonstrate compliance.

2.11.3 From heuristics to primal–dual governance signals

Most early-stage governed strategies rely on heuristics: overlays, stop-outs, penalty weights, and ad hoc safety rules. These are useful, but they often lack a principled way to quantify the pressure constraints exert on the policy. Primal–dual methods provide that principled layer. In primal–dual optimization, the policy parameters are the primal variables (the decisions), while multipliers associated with constraints are the dual variables (the governance shadow prices). The dual variables measure, in a precise way, how costly it is to satisfy each constraint.

This becomes an institutional upgrade because dual variables are interpretable governance signals. If the multiplier on turnover is persistently high, the policy is "wanting" to churn more than allowed. If the multiplier on drawdown risk rises sharply under certain regimes, it indicates that the policy's mechanism becomes incompatible with drawdown governance in those regimes. These are not vague warnings; they are quantitative signals that can be logged, reviewed, and used for escalation. They also provide a systematic way to tune constraint enforcement without arbitrary weight fiddling: dual variables adapt to enforce constraints rather than relying on fixed penalty weights chosen by intuition.

In Chapter 3, this primal–dual logic will be translated into an auditable training loop: constraint costs are measured, multipliers are updated, and the policy is adjusted to improve reward while respecting constraints. The result is a controller that is not merely filtered by governance after training, but shaped by governance during training.

2.11.4 From single-run success to reproducible acceptance

Another reason robustness testing is not enough is that passing a stress suite once is not acceptance. Institutions do not accept single-run success because single-run success can be luck, selection bias, or an artifact of a particular simulation path. Acceptance must be reproducible across seeds, across scenario packs, and across evaluation windows. Chapter 2 introduced cross-seed validation and stress testing, but Chapter 3 will elevate this into a formal acceptance protocol tied to constrained learning.

In a constrained control pipeline, reproducible acceptance means more than "performance repeats." It means constraint compliance repeats, boundary time remains controlled, and tail behavior remains bounded across many independent draws. It also means the training process itself is reproducible: given the same run manifest and seeds, the same policy is obtained. This is an institutional requirement because it enables audit, regression testing, and controlled deployment. Chapter 3 will therefore treat reproducibility as part of the controller specification: training, evaluation, and artifact generation become standardized, versioned processes, not one-off experiments.

2.11.5 Chapter 3 promise: a governed surrogate controller pipeline

Chapter 3's promise is the conversion of a surrogate agent program into a constrained, supervised controller pipeline. It will produce a controller that is trained (or tuned) within a CMDP framing, with explicit constraints that reflect institutional mandates. It will produce primal–dual governance signals as first-class artifacts: multipliers, binding-constraint diagnostics, and stability measures across training. It will integrate scenario packs not only as evaluation tools but as training regularizers: domain randomization and stress-aware learning that reduces sensitivity to model error.

Most importantly, Chapter 3 will produce an institutional deliverable: a governed surrogate controller pipeline that can be replayed, audited, and reviewed by a committee. The output is not merely a function that maps state to action. It is a function embedded in a governance system: typed configurations, deterministic run identities, constraint compliance evidence, and escalation triggers. This is the upgrade from "we optimized a proxy and tested it" to "we trained a constraint-respecting controller under supervision, with reproducible acceptance criteria." That is what makes the surrogate agent program institutional rather than experimental.

Bibliography

[1] Eitan Altman. *Constrained Markov Decision Processes.* Chapman & Hall/CRC, 1999.

[2] Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming.* Wiley, 1994.

[3] Joshua Achiam, David Held, Aviv Tamar, and Pieter Abbeel. Constrained Policy Optimization. In *Proceedings of the 34th International Conference on Machine Learning (ICML)*, 2017.

[4] Yinlam Chow, Mohammad Ghavamzadeh, Lucas Janson, and Marco Pavone. Risk-Constrained Reinforcement Learning with Percentile Risk Criteria. *Journal of Machine Learning Research*, 18(167):1–51, 2017.

[5] Chen Tessler, Daniel J. Mankowitz, and Shie Mannor. Reward Constrained Policy Optimization. In *International Conference on Learning Representations (ICLR)*, 2019.

[6] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction.* MIT Press, 2nd edition, 2018.

[7] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization.* Cambridge University Press, 2004.

[8] Dimitris Bertsimas and Melvyn Sim. The price of robustness. *Operations Research*, 52(1):35–53, 2004.

[9] R. Tyrrell Rockafellar and Stanislav Uryasev. Optimization of conditional value-at-risk. *Journal of Risk*, 2(3):21–41, 2000.

[10] Robert Almgren and Neil Chriss. Optimal execution of portfolio transactions. *Journal of Risk*, 3(2):5–39, 2001.

Chapter 3

Constrained Surrogate Controllers: CMDPs, Primal–Dual Governance, and Stage Gates

Abstract. Chapter 3 completes the institutional escalation of surrogate trading agents by replacing penalty-driven proxy optimization with *constrained control*. The central claim is practical: robustness testing after optimization is necessary but insufficient when an optimizer can trade constraint violations against reward. Institutional mandates—drawdown limits, turnover caps, tail-risk budgets, concentration limits, and feasibility requirements under liquidity stress—must be enforced during learning, not merely checked after the fact. We therefore formulate the surrogate agent as a *Constrained Markov Decision Process (CMDP)* in which performance is optimized subject to explicit risk and execution constraints that are treated as first-class design variables.

The chapter introduces a governed primal–dual training framework. Policy parameters define the primal decision system (how the agent acts), while Lagrange multipliers define the dual governance system (how strongly each constraint is enforced). These multipliers are interpreted as *shadow prices* that quantify institutional pressure: they reveal which constraints are binding, when the agent "wants" to violate limits, and how constraint tension changes across regimes and stress. This produces a new class of audit artifacts: constraint-cost trajectories, multiplier stability diagnostics, boundary-time profiles, and reproducible acceptance evidence across multiple deterministic seeds and scenario packs.

Robustness is integrated into the training loop through domain randomization and stress-aware sampling. The agent is trained across a distribution of synthetic environments that vary regime prevalence, switching rates, liquidity conditions, and tail-event intensity, ensuring that constraint compliance is not a fragile property of a single world. The chapter defines stage-gate acceptance as a distributional standard: near-zero breaches, controlled boundary time, bounded CVaR and drawdown under stress, and stable performance relative to conservative and MPC baselines across held-out seeds. The deliverable is not merely a high-scoring strategy, but a *governed surrogate controller pipeline*: reproducible, auditable, supervision-ready, and explicitly designed to resist Goodhart failures under optimization pressure.

3.1 Chapter Purpose: The Institutional Upgrade from Proxy Optimization to Constrained Control

3.1.1 Why Chapter 3 exists: penalties do not enforce mandates

Chapter 3 exists because institutions do not run trading systems as "maximize reward with some soft preferences." They run them as supervised control systems with mandates. A mandate is not a suggestion. It is an operational boundary that, if crossed, triggers escalation: de-risking, human review, capital withdrawal, or shutdown. Penalty-based proxy objectives struggle in this setting because penalties create a negotiable exchange rate between performance and violation. An optimizer will discover that exchange rate and exploit it. If the penalty is finite, then the optimizer can decide that violating a constraint is worth it. Even if the penalty is large, the optimizer can still seek boundary states where it earns performance by living "just inside" the constraint, increasing fragility. These behaviors are not anomalies; they are expected once optimization pressure is applied.

Chapter 2 showed how these failure patterns reveal themselves: tail selling disguised as stability, churn driven by weak cost modeling, boundary seeking under leverage and turnover constraints, and brittle regime conditioning that breaks under distribution shift. Stress testing and diagnostics can detect these behaviors, and that detection is valuable. But if the optimization process remains penalty-driven, the system can keep generating exploitative candidates. The pipeline becomes reactive: train, test, reject, revise weights, repeat. This is not how institutions reach production readiness. They need an upgrade that changes the optimization problem itself so that constraint compliance is part of the feasible set, not a post-hoc filter.

This is the institutional logic behind constrained control. Instead of hoping that penalties will shape behavior, we impose constraints that reflect mandates. The agent is then asked to maximize performance within those constraints. This is not a philosophical preference; it is an engineering requirement. In real deployment, the institution's risk systems enforce constraints in real time: pre-trade checks, leverage caps, concentration limits, and drawdown stop-outs. A learning system that cannot operate within such constraints is not deployable. Therefore, Chapter 3 focuses on the design of a controller that is trained under constraint enforcement and on the governance signals that prove that enforcement is stable.

The most important consequence of the shift is that "good" becomes redefined. Under penalty optimization, a "good" policy can be one that slightly violates constraints in exchange for higher reward, or that spends most of its time near limits. Under constrained optimization, "good" means the policy stays within limits with buffer, including under stress and model error, and still produces acceptable performance. This aligns with institutional supervision: the institution is willing to accept lower headline returns for higher reliability, because reliability is what makes capital allocation possible at scale.

Finally, the shift resolves a governance contradiction that penalty systems create. In penalty systems, governance settings (like drawdown limits) become tunable knobs. Researchers can implicitly or explicitly tune them to improve results, undermining supervision. In constrained control, governance settings are fixed as part of the contract. Learning happens under them, not through them. This preserves the institutional separation between "what the institution requires" and "what the model learns."

3.1.2 What "constraint-respecting" means in trading systems

Constraint-respecting is not a buzzword; it is a concrete behavioral specification. In trading systems, constraints are multi-dimensional and state-dependent. A constraint-respecting controller is one that maintains compliance across these dimensions, not only on average but across time, stress states, and measurement uncertainty.

First, constraint-respecting means respecting hard limits with near-zero breach probability. For example, a leverage cap is typically hard. A policy that occasionally breaches leverage is not acceptable, regardless of average performance. Similarly, turnover constraints often reflect operational limits. A policy that occasionally spikes turnover beyond capacity is not acceptable because those spikes tend to occur in precisely the worst states: transitions and stress. Drawdown constraints can be both hard and procedural: crossing them triggers governance escalation. A constraint-respecting controller must treat drawdown triggers as first-class signals, not as minor penalties.

Second, constraint-respecting means maintaining buffers. Institutions do not operate at the exact boundary because real systems have delays, slippage, and measurement error. Therefore, a robust controller must spend limited time near boundaries. Boundary time becomes an explicit acceptance target. This is one of the most important translations from mathematics to governance: compliance is not only "no breaches," it is "no boundary living." A controller that lives at the boundary is fragile and likely to fail under small perturbations.

Third, constraint-respecting means recognizing that constraints are often state-dependent. Liquidity limits tighten in stress. Volatility budgets shrink when volatility rises. Diversification constraints become more binding when correlation spikes. A controller that is compliant in calm regimes but violates feasibility under stress is not truly constraint-respecting. Therefore, a governed controller must incorporate state-dependent scaling or overlays: reduce exposure when volatility rises, reduce trading when liquidity deteriorates, and reduce concentration when correlation compresses. These behaviors are part of what institutions mean by "risk management," and in this chapter they become part of the control policy itself.

Fourth, constraint-respecting means making constraints observable and auditable. The controller must output not only actions but constraint diagnostics: estimated costs, estimated risk budgets, and margin to limits. If the controller is a black box that cannot report these values, supervision becomes impossible. Therefore, Chapter 3 treats constraint signals as part of the state representation and treats constraint margins as part of the artifact bundle.

Fifth, constraint-respecting means stability under optimization. Some training methods can satisfy constraints in expectation while oscillating violently in behavior, especially when constraint enforcement is implemented via multipliers that update too aggressively. An institution cannot deploy a controller that oscillates between risk-on and risk-off regimes due to training instability. Therefore, constraint-respecting also includes stable multiplier dynamics, stable policy updates, and stable behavior profiles.

These requirements make clear why Chapter 3 cannot be simply "add bigger penalties." Bigger penalties do not guarantee buffers, do not guarantee state-dependent compliance, and do not guarantee stable enforcement. Constrained control is not about making penalties large; it is about changing the optimization problem so that the controller must learn to operate within the institution's feasible region.

3.1.3 How Chapter 3 builds directly on Chapter 2 diagnostics and stage gates

Chapter 2 produced evidence. It identified exploitation patterns and measured robustness gaps. Chapter 3 uses that evidence as design input. The transition is not arbitrary; it is a methodological escalation grounded in observed failure modes.

If Chapter 2 evidence shows tail selling, Chapter 3 responds by turning tail risk into a constraint signal, such as a CVaR budget or a worst-case loss budget under scenario packs. Instead of hoping that a tail penalty prevents tail selling, Chapter 3 enforces tail constraints during learning. If Chapter 2 evidence shows churn exploitation, Chapter 3 turns turnover and participation into hard constraints and integrates convex cost modeling into the learning loop so that the controller learns that trading is expensive and limited. If Chapter 2 evidence shows boundary living, Chapter 3 introduces buffer constraints or explicit boundary-time penalties that are treated as governance metrics, not merely as soft preferences.

Chapter 2 also established the governance infrastructure: run identities, deterministic seeds, scenario packs, evaluation suites, baseline comparators, and artifact bundles. Chapter 3 reuses this infrastructure but changes what is optimized. This is critical for institutional continuity. The evaluation battery remains the same, allowing comparability across chapters. What changes is that the training objective and the controller architecture are upgraded so that passing the evaluation is not an accident but a consequence of constraint-respecting learning.

The stage-gate logic also carries over. Chapter 2 ended with "advance" or "do not advance" decisions based on evidence. Chapter 3 strengthens this by introducing acceptance protocols that are distributional: passing across many seeds and many stress configurations, with near-zero breaches and limited boundary time. Chapter 3 therefore does not replace governance; it deepens it. The goal is to make the system more predictable and more defensible.

A particularly important carry-over is the idea of proxy-critical parameters. Chapter 2 identified which assumptions—cost convexity, stress prevalence, regime switching rates—were critical to acceptability. Chapter 3 uses this to design domain randomization and robust training. Instead of training in one world and hoping to generalize, Chapter 3 trains across a distribution of worlds that spans plausible ranges of these critical parameters. This reduces sensitivity to model error and reduces the likelihood that the controller is optimized for a narrow environment.

In summary, Chapter 3 is not a reset. It is a response. It takes the diagnostic truth-telling from Chapter 2 and turns it into a design upgrade: constraints are enforced during optimization, robustness is injected into training, and acceptance is defined as reproducible compliance.

3.1.4 What the controller is (and is not): a governed policy object

A central institutional requirement is clarity about what has been produced. Chapter 3 produces a controller that is an object with a specific interface, a specific governance contract, and specific audit artifacts. It is not “an AI that trades.” It is not “a model that predicts returns.” It is a function (or function class) embedded in a constraint system and tied to an auditable run identity.

At minimum, the controller includes: (i) a state representation that includes belief-state features and constraint-relevant features, (ii) a policy mapping from state to action, (iii) a constraint registry defining limits and budgets, (iv) a governance overlay that enforces hard constraints and applies stop-out logic, and (v) an artifact bundle that records configuration, training logs, evaluation results, and acceptance status. The controller is therefore a composite object: policy plus governance.

What the controller is not is equally important. It is not a guarantee of profitability. It is not a claim about real markets. It is a surrogate controller trained in a synthetic environment, intended as a disciplined step toward supervised deployment. Its purpose is to demonstrate that constraint-respecting learning can produce controlled behavior under optimization pressure. The controller should therefore be described with “Not verified” language and should require human review and sign-off.

The controller is also not an opaque black box. Even if the policy class is nonlinear, the controller must expose governance signals: constraint costs, multipliers, boundary margins, and breach flags. In institutional terms, the controller must be monitorable. If it cannot be monitored, it cannot be deployed.

Finally, the controller is not trained once and forgotten. The institutional view is that controllers are versioned artifacts. Each version has a run identity and acceptance evidence. Changes require re-evaluation and re-approval. Therefore, Chapter 3 defines not only the controller but also the change management process by which controllers are revised.

3.1.5 Institutional success criteria for Chapter 3

The success criteria of Chapter 3 are designed to be committee-facing and operational, not merely academic. They define what it means for the constrained control upgrade to have worked.

First, constraint compliance must be strong and reproducible. Breach rates should be near zero across held-out seeds and stress scenarios. Boundary time should remain below defined thresholds. The controller should show conservative buffers rather than living at limits. This is the core institutional requirement.

Second, constraint enforcement must be stable. Primal–dual multipliers should not explode or oscillate. Policy behavior should not exhibit unstable toggling between risk-on and risk-off purely due to enforcement instability. The training process should produce smooth convergence in constraint satisfaction and stable policy profiles.

Third, performance must be acceptable relative to baselines. The institution is not seeking maximum theoretical return; it is seeking feasible return under governance. Therefore, the controller should outperform conservative baselines net of costs while maintaining a clean diagnostic profile. If it cannot, the institution may prefer the baseline and consider the constrained learning unnecessary.

Fourth, robustness must improve relative to Chapter 2 candidates. Under scenario packs and distribution shifts, the constrained controller should fail gracefully: reduce risk, reduce turnover, and remain within constraints. The goal is not that performance never degrades, but that degradation remains within acceptable boundaries.

Fifth, audit artifacts must be complete. The notebook must export run manifests, configuration hashes, environment fingerprints, training logs including multiplier trajectories, evaluation summaries, and stage-gate acceptance decisions. These artifacts are not administrative overhead; they are the evidence that makes the controller institutional.

Finally, Chapter 3 success is demonstrated by the ability to make an institutional claim with precision: "We obtained a constraint-respecting surrogate controller object that maps a bounded state representation to bounded actions while satisfying explicit governance constraints under a distribution of synthetic stress conditions, with reproducible acceptance evidence and audit artifacts." This is the level of specificity required for production readiness.

3.1.6 Synthesis: why this section is the conceptual gate

This first section is the conceptual gate for Chapter 3. It explains why constrained control is not an optional refinement but the institutional upgrade required to move from "we tested proxies under optimization" to "we train controllers under mandates." It clarifies what constraint-respecting means, how Chapter 3 builds on Chapter 2, what the controller object is, and what success looks like in institutional terms.

With this framing established, the next sections can become precise: we can write the CMDP formulation, define constraint signals, implement primal–dual training, integrate robustness into training, and define acceptance protocols. But none of those technical steps matter if the institutional purpose is not clear. Chapter 3 begins with this purpose because purpose determines what must be enforced, what must be logged, and what must be accepted. In constrained surrogate control, governance is not a wrapper. It is the design.

3.2 CMDP Formulation for Trading Surrogate Agents

3.2.1 State, action, transition: the controlled market abstraction

A constrained surrogate trading agent is, at its core, a control system embedded in a stochastic environment. To make the problem governable, we must define precisely what the agent observes, what it can do, and how the world evolves in response. This is the purpose of the Markov Decision Process abstraction: to define a state space, an action space, and a transition mechanism. However, because the financial reality is partially observed and execution is endogenous, we must be disciplined about what "state" means. In Chapter 3, we use a controlled market abstraction that separates latent reality from operational observables, and we treat this separation as a governance choice, not a convenience.

Let the environment evolve in discrete time steps $t = 0, 1, 2, \ldots$ corresponding to a decision cadence (e.g., daily, weekly). The agent does not observe the full latent market state. Instead, it observes an operational state vector s_t constructed from (i) belief-state features derived from filters and estimators, (ii) execution and feasibility features that summarize current trading conditions, and (iii) governance-relevant summaries of current portfolio exposure and constraint margins. This choice is intentional: a controller cannot be supervision-ready if it does not "see" the variables that supervision cares about.

The action a_t is the control decision. In a surrogate trading agent, actions can be defined at different levels: target exposure, target portfolio weights, or a discrete action set representing allowable rebalancing moves. Institutional governance prefers bounded action spaces because bounded actions make feasibility easier to enforce. Therefore, a typical action definition is a bounded target exposure vector Δw_t or a bounded target weight vector w_t subject to caps, turnover limits, and leverage limits. Alternatively, in a simplified controlled laboratory, the action can be a scalar exposure level to a synthetic factor portfolio, with a discrete set of allowed values. The abstraction can scale from simple to complex, but the principle remains: actions must be bounded and interpretable in terms of risk and feasibility.

The transition kernel $P(s_{t+1} \mid s_t, a_t)$ captures how the operational state evolves. In finance, actions influence not only portfolio returns but also future feasibility. Trading changes positions and induces execution costs; high turnover can reduce future flexibility by consuming risk budgets, triggering governance overlays, or increasing drawdown. Therefore, the transition must include portfolio dynamics: holdings update, costs realize, P&L updates wealth, risk measures update, and belief-state filters update based on new observations. This creates a closed-loop system where action affects future state.

The Markov property is a modeling commitment: we construct s_t to contain enough information so that the future distribution depends on the past only through s_t and a_t. In practice, this is an approximation. Institutional governance does not demand perfect Markov structure; it demands that the state representation be bounded, interpretable, and sufficient to enforce constraints and produce stable behavior. The design philosophy is: include what matters for constraints and stability; exclude what invites leakage or ungovernable complexity.

Finally, because the environment is synthetic, we can define the transition mechanism explicitly and perturb it. This is crucial for constrained learning. In Chapter 3, we do not train in one fixed transition kernel. We train in a family of kernels parameterized by regime structures, liquidity conditions, and tail-event intensity. This creates a distribution over transitions that models uncertainty and supports robust constrained learning.

3.2.2 Reward versus costs: separating performance from governance

A standard MDP defines a reward function $r(s_t, a_t)$ that the agent maximizes. In trading, it is tempting to fold everything into reward by subtracting penalties and costs. Chapter 3 rejects this temptation as an institutional risk. The core upgrade is to separate performance from governance: define a reward that represents economic objective (e.g., risk-adjusted return) and define separate cost signals that represent governance constraints (e.g., drawdown cost, turnover cost, tail-risk cost). This separation is the foundation of the CMDP.

The reward signal should represent what the institution wants to optimize *within* mandates. In a surrogate lab, reward can be defined as net P&L, or as a utility function of net returns. However, purely maximizing net P&L is often inconsistent with institutional mandates because it can encourage high variance and tail exposure. Therefore, reward is often defined as a concave utility such as expected return minus a modest variance penalty, or as a normalized return measure. The key is that reward is not the place where hard governance rules live. Reward expresses the objective; constraints express the mandates.

Cost signals are then defined for each constraint dimension. For example, define a turnover cost signal c_t^{to} representing realized turnover or participation. Define a drawdown cost signal c_t^{dd} representing drawdown increment or an indicator of drawdown exceeding a threshold. Define a tail-risk cost signal c_t^{tail} representing contributions to tail loss, such as negative returns beyond a quantile or scenario-based worst-case losses. Define concentration costs c_t^{conc} capturing max weight or effective number of names. These cost signals are not penalties to be traded; they are quantities to be constrained.

This separation has a profound governance benefit: the institution can specify constraint thresholds independently of reward tuning. In penalty-based systems, changing a penalty weight changes the effective tolerance for violations. In a CMDP, tolerances are explicit: each cost has a budget d_i such that the expected cumulative cost must remain below d_i. This aligns with mandates: "expected turnover must remain below X," "expected tail loss must remain below Y," "probability of drawdown breach must be below Z." The specific mathematical form of the constraint depends on the mandate, but the conceptual separation is the same.

Another key benefit is interpretability. When reward and costs are separate, we can log them separately. We can see when performance is achieved by increasing costs, which is often unacceptable. We can identify which constraints are binding, and we can interpret dual variables as shadow prices. This is the institutional logic of constraint-respecting learning: it makes trade-offs visible and governable.

3.2.3 Hard limits, soft limits, and risk budgets

Institutions do not treat all constraints equally. Some are hard limits: leverage caps, regulatory restrictions, and operational ceilings. Others are risk budgets: targets that can be approached but should not be exceeded in expectation, such as volatility targets or average turnover budgets. Still others are escalation triggers: drawdown levels that trigger review rather than immediate shutdown. A CMDP formulation must encode these distinctions explicitly, because they determine how learning and enforcement are structured.

Hard limits are often enforced at the action level. For example, leverage cap can be enforced by projecting any proposed action onto the feasible action set. Turnover cap can be enforced by clipping rebalancing moves. Concentration cap can be enforced by capping weights. These enforcement mechanisms are deterministic and should produce zero breaches by construction. In a CMDP, hard limits can still be represented as constraints, but operationally it is safer to enforce them as action-space restrictions. This prevents learning from ever exploring infeasible actions, reducing risk and improving stability.

Soft limits and budgets are typically enforced as expected constraints. For example, one might tolerate occasional high turnover if overall turnover remains within a budget, or one might tolerate small drawdowns while enforcing a hard stop at a larger drawdown. In CMDP terms, these are constraints on expected cumulative costs. Enforcement can be achieved via primal–dual methods: dual variables adjust to keep expected costs within budgets. This is where learning enters: the policy learns to trade off reward against costs within the feasible region defined by budgets.

Escalation triggers require special handling because they represent governance procedures, not merely costs. A drawdown trigger might require the policy to de-risk or to stop trading. In a surrogate lab, we can encode escalation as a state transition: once drawdown exceeds a threshold, the environment enters a “review state” where actions are restricted or forced to zero exposure. This models institutional reality: after certain triggers, autonomy is reduced. Encoding escalation this way ensures the controller learns under realistic governance.

The distinction between hard and soft constraints also determines what is “learned” versus what is “fixed.” Hard limits are typically fixed and enforced by projection. Soft limits can be enforced by adaptive multipliers. The governance decision is: which constraints are negotiable budgets and which are mandates. Chapter 3 treats this as a contract design problem. The contract must be stable and defensible, and it must not allow optimization to negotiate away mandates.

3.2.4 Constraint feasibility under partial observability and execution

A key difficulty in trading CMDPs is that constraints depend on quantities that are not perfectly observable and that are affected by execution. For example, the true realized slippage is uncertain when the action is chosen. Future volatility is unknown. Liquidity can deteriorate unexpectedly.

Therefore, feasibility is not a static set; it is a stochastic concept. A policy can choose an action that appears feasible given current estimates but becomes infeasible when execution costs realize or when the market moves.

Institutional governance responds by building buffers and by enforcing constraints in conservative ways. In the CMDP, this translates into conservative cost estimators and robust constraint definitions. For example, instead of constraining expected turnover alone, one might constrain a high quantile of turnover under stress. Instead of constraining expected drawdown, one might constrain worst-case drawdown across scenario packs. Instead of constraining expected tail loss, one might constrain CVaR. These are ways of making feasibility robust to uncertainty.

Partial observability also motivates belief-state representation. Constraints often depend on latent regimes: volatility regime, liquidity regime, correlation regime. The agent cannot observe these directly, but it can maintain beliefs. The state s_t therefore includes belief features, and constraints can depend on these beliefs. For example, when crisis belief rises, risk budgets tighten. This creates an institutionally realistic dynamic: governance becomes stricter when uncertainty or stress rises. The CMDP can encode this by making constraint budgets state-dependent functions $d_i(s_t)$, or by adding governance overlay rules conditioned on belief.

Execution makes feasibility endogenous because trading affects costs and future state. High turnover increases costs and can amplify drawdowns. High participation can steepen impact. Therefore, the CMDP must include execution in the transition and in cost signals. Costs are not external penalties; they are part of the system dynamics. A constraint-respecting policy must learn that aggressive trading reduces future feasibility and increases risk of breach. This feedback loop is one of the reasons constrained learning is necessary: the policy must internalize that feasibility is a dynamic resource.

In institutional terms, this subsection clarifies why "constraints are hard" does not mean "constraints are simple." Hard constraints in finance require conservative estimation, buffers, and state-dependent tightening. Chapter 3's methodology is to encode these mechanisms explicitly so that the learned controller behaves like an institutional system: cautious under uncertainty, aggressive only when feasible, and always within supervised boundaries.

3.2.5 CMDP acceptance: why feasibility is the first theorem

Before we discuss algorithms, we must clarify what it means for the CMDP to be solvable in an institutional sense. In theory, a CMDP can be infeasible: there may be no policy that satisfies all constraints while achieving nontrivial reward. In practice, this is common if constraints are too tight, if the environment is too adversarial, or if the action space is too limited. Institutions face analogous problems: some mandates cannot be satisfied simultaneously with certain return targets. Therefore, feasibility is not guaranteed; it must be checked and documented.

This is why “feasibility is the first theorem.” Chapter 3’s pipeline must include a feasibility analysis stage that answers: does there exist a policy in the chosen class that satisfies constraints across the scenario distribution? We do not need a formal proof, but we need evidence. This evidence can come from baselines: if conservative controllers satisfy constraints, feasibility is plausible. It can come from initial training runs: if primal–dual training converges to constraint satisfaction without collapsing reward to zero, feasibility is plausible. If no policies satisfy constraints without going flat, then constraints or environment must be revised.

Feasibility also defines acceptance. An institution cannot accept a controller that satisfies constraints only by taking no risk and producing no activity, unless that is explicitly desired. Therefore, acceptance requires both constraint compliance and meaningful operation. This is a “constrained performance” standard: achieve at least baseline-level reward while satisfying constraints. Chapter 3 will formalize this through acceptance thresholds relative to baselines and through stage-gate criteria.

From a governance viewpoint, feasibility analysis prevents a common failure mode: blaming the algorithm for what is actually a contract design problem. If constraints are inconsistent, no algorithm will produce an acceptable policy. Therefore, Chapter 3 treats CMDP formulation as contract design, and it treats feasibility as the first acceptance check.

3.2.6 Synthesis: the CMDP framing as the mathematical backbone of governed control

This section provides the mathematical backbone of Chapter 3’s institutional upgrade. It defines the controlled abstraction (state, action, transition), separates reward from governance costs, distinguishes hard limits from budgets, and addresses feasibility under partial observability and execution. The outcome is a CMDP framing that aligns with how institutions think: optimize performance within mandates, under uncertainty, with conservative buffers and auditable signals.

Once this framing is in place, we can proceed to the next section, which specifies the constraint signals in detail: drawdown, turnover, tail risk, concentration, and latency-aware enforcement. Those signals are the operational encoding of mandates. The CMDP defines how they enter the learning problem; the constraint registry defines what they are.

sectionConstraint Signals: What Institutions Actually Control

3.2.7 Drawdown budgets as stop conditions and review triggers

Drawdown is the governance variable most closely associated with institutional escalation. Unlike volatility, which can be tolerated if it is understood and budgeted, drawdown is experienced as a failure of control: capital has been lost, confidence is impaired, and stakeholders demand explanation. For that reason, drawdown is rarely treated as a mere metric. It is treated as a procedural trigger.

Chapter 3 therefore encodes drawdown not only as a cost signal but as a stateful governance mechanism that can change what the controller is allowed to do.

To model drawdown in a surrogate trading controller, define the portfolio value process V_t net of execution costs. Define the running peak $M_t = \max_{0 \le k \le t} V_k$. Then drawdown is $D_t = 1 - V_t/M_t$. This quantity is simple but its governance interpretation is subtle: drawdown is not just "how much we lost," but "how far we are from a high-water mark," which aligns with institutional memory and investor perception.

Institutions typically implement drawdown in layers. There is often a soft drawdown budget (a target range where risk should be reduced) and a hard stop-out threshold (where the system must de-risk immediately and trigger review). The surrogate controller should be designed to reflect this. In CMDP terms, we can define a drawdown cost signal c_t^{dd} that increases as D_t rises, and we can impose a budget d_{dd} on expected cumulative drawdown cost. But more importantly, we can encode a stop-out as a hard transition: if D_t exceeds D_{hard}, the environment enters a "review regime" where the action space is restricted (for example, only allowing reductions in exposure) or even forces the policy to go flat.

This encoding has three institutional advantages. First, it makes drawdown a first-class state variable: the policy observes it and can react. Second, it enforces the procedural truth: after a hard drawdown event, autonomy is reduced. Third, it prevents reward hacking where a policy trades occasional catastrophic drawdowns against long periods of small gains. If hard stop-out resets the episode or imposes significant operational consequences, tail selling becomes less attractive.

In addition to thresholds, institutions care about drawdown *duration*. A long drawdown can be worse than a sharp drawdown because it erodes confidence and can trigger risk committee intervention. Therefore, the constraint system can include a duration signal: the time spent with D_t above a soft level. This creates a dynamic constraint that discourages policies that linger in impaired states. In practical controlled designs, duration constraints can be implemented as a cost that accumulates when drawdown is above a soft threshold.

Finally, drawdown constraints must be evaluated under stress scenarios and under cost inflation. Drawdowns are often amplified by execution and by correlation spikes. A policy that appears drawdown-safe in calm conditions can fail under stress. Therefore, drawdown budgets are not only constraints; they are acceptance metrics across scenario packs. Chapter 3 encodes this by logging drawdown trajectories per scenario, computing worst-case drawdown across the scenario pack, and enforcing drawdown stop-outs consistently.

3.2.8 Turnover, participation, and operational feasibility limits

Turnover is the operational heartbeat of a trading system. High turnover is not merely "more active"; it is a claim about capacity, infrastructure, and execution quality. Institutions have finite operational bandwidth: order routing, monitoring, reconciliation, and error management. Even if a strategy is theoretically profitable, excessive turnover can make it infeasible and can magnify model risk through execution sensitivity. Therefore, turnover is one of the most common institutional constraints, and it is one of the most frequently exploited proxy surfaces when it is not enforced as a mandate.

Turnover can be defined as $T_t = \sum_i |w_{t,i} - w_{t-1,i}|$ for portfolio weights, or as absolute traded notional relative to capital. Participation extends turnover into microstructure: it measures the fraction of market volume consumed by the strategy. In synthetic labs, we may not model full order books, but we can still model a proxy for participation by relating traded notional to an available liquidity depth parameter. The key institutional idea is capacity cliffs: as participation rises, impact costs rise nonlinearly and fills become worse. Therefore, participation is not only a constraint; it is a feasibility boundary.

Chapter 3 treats turnover and participation constraints in two complementary ways. First, as action-space restrictions: rebalancing moves are capped, ensuring that the controller cannot propose turnover beyond operational limits. This is a strong form of hard constraint enforcement. Second, as cost signals in the CMDP: turnover contributes to cost and is constrained in expectation, enabling the controller to learn smoother behavior that respects budgets even when it is not forced by hard caps.

These constraints must be state-dependent. In stress, liquidity collapses, and the same turnover is more expensive and more disruptive. Institutions often tighten turnover budgets under stress: "trade less when markets are thin." Therefore, Chapter 3 encodes turnover budgets that tighten when volatility or liquidity stress signals rise. Operationally, this can be done through a liquidity multiplier that scales effective turnover caps. The controller observes this multiplier and learns to reduce trading in stress.

Turnover constraints also require buffer enforcement. A policy that consistently trades at the turnover cap is fragile: small slippage or measurement noise can push it over. Therefore, Chapter 3's acceptance criteria include turnover boundary time and turnover margin distributions. A policy must not only respect turnover caps; it must do so with margin. This is an explicitly institutional target.

Finally, turnover and participation constraints must be integrated with the execution model. If the cost model is convex in participation, then high turnover becomes automatically unattractive, but only if the convexity is realistic and conservative. Chapter 3 therefore treats convex impact as part of constraint design, not merely as a cost term. If impact is under-modeled, turnover constraints become the primary defense. If impact is conservatively modeled, constraints and costs reinforce each other.

3.2.9 Tail risk budgets: CVaR and stress-conditional limits

Tail risk is where institutions lose careers. Most catastrophic trading failures are tail events: extreme losses under stress, correlation spikes, and liquidity collapses. Tail risk is also where proxy objectives are most likely to be hacked, because tail events are rare and easy to ignore in finite samples. Therefore, an institutional surrogate controller must treat tail risk as a constrained budget, not as a secondary metric.

A practical institutional tail budget is often expressed through Conditional Value-at-Risk (CVaR) or Expected Shortfall. CVaR measures the expected loss given that we are in the worst $q\%$ of outcomes. In a sequential decision problem, we must be careful about what distribution we apply CVaR to. We can compute CVaR of one-step returns, or of cumulative episode returns, or of drawdown increments. Each choice corresponds to a different governance view. Institutions often care about drawdowns and cumulative loss events, so CVaR of cumulative loss over a horizon can be more relevant.

In a surrogate lab, we have a powerful tool: scenario packs. Instead of estimating tail risk from rare events in one path, we can generate tail scenarios deliberately. This allows us to define stress-conditional tail budgets. For example, we can define CVaR across scenarios: compute worst-case outcomes across a scenario pack and then constrain the tail of that distribution. Alternatively, we can define a tail cost signal that activates under certain stress regimes (high volatility, correlation spike) and constrain its expectation. The key principle is that tail risk must be visible during learning. If tail events are absent from training, the controller can learn behavior that is safe only because it never saw tails.

Tail budgets can also be implemented through worst-case constraints: "worst-case loss across the stress suite must be above $-L$." Worst-case constraints are conservative and can be too strict, but they are often institutionally appealing because they align with safety-first mandates. A more flexible approach is CVaR across stress scenarios, which allows some tail outcomes but controls the average of the worst ones.

In addition to losses, tail risk includes liquidity and correlation co-movements. Tail constraints should therefore be conditional: when stress indicators rise, risk budgets tighten. A surrogate controller can implement this through state-dependent constraints: $d_{\text{tail}}(s_t)$ shrinks in high-volatility or low-liquidity states. This models the institutional reality that risk committees reduce risk budgets in crises.

Finally, tail constraints must be auditable. The controller must output tail-risk estimates, the scenario outcomes used to compute them, and the constraint margins. If tail risk is computed from scenario packs, the scenario definitions must be versioned and logged. This transforms tail risk from a vague concern into a measurable, reviewable mandate.

3.2.10 Concentration and correlation exposure limits

Diversification is a constraint, not a slogan. In calm markets, portfolios can look diversified because correlations are moderate and dispersion exists. In stress, correlations rise, dispersion collapses, and portfolios become concentrated in effective risk. Institutions therefore impose concentration limits and factor exposure limits to prevent strategies from becoming hidden one-factor bets. These limits are essential in surrogate agents because optimization can easily discover concentrated exposures that look good under baseline but fail under correlation stress.

Concentration can be defined in multiple ways: maximum weight per asset, Herfindahl index of weights, effective number of names, sector or factor exposures, and concentration of risk contributions. In a synthetic lab with factor structures, concentration constraints can be defined on exposures to synthetic factors. For example, limit exposure to a "market" factor and to a "crisis" factor. This captures the institutional intent: avoid portfolios that are effectively one big bet.

Correlation exposure limits are more subtle because correlation is state-dependent. One approach is to include correlation stress scenarios in the scenario pack and require that the portfolio remain within concentration limits under those scenarios. Another approach is to include an online estimate of correlation structure in the state and define a constraint on effective diversification conditional on correlation. For example, require that effective number of bets remains above a threshold. In practice, in a no-pandas, pure-Python lab, this can be implemented with rolling covariance estimators and eigenvalue summaries. The key is to keep representation bounded and interpretable.

Institutions also care about concentration dynamics: not just average concentration but peaks. A policy may rotate across assets and appear diversified over time, but at each moment it can be highly concentrated. Therefore, constraints should be enforced point-in-time. This is consistent with operational risk: a shock at a moment of concentration can cause large loss. Thus, the controller's action projection must cap weights pointwise, and evaluation must report maximum concentration over time.

Concentration constraints interact with turnover constraints. Tight concentration caps can force diversification, but they can also increase turnover as the policy rebalances to maintain diversification. The CMDP formulation allows us to handle this trade-off explicitly: turnover and concentration are separate constraint signals, and primal–dual learning can find policies that satisfy both. This is one of the advantages of constrained learning: it makes multi-constraint trade-offs explicit rather than implicit.

Finally, concentration constraints must be stress-tested under correlation compression. A policy that respects concentration in normal correlations may become concentrated in effective risk when correlations rise. Therefore, Chapter 3 includes correlation stress in training via domain randomization and in acceptance via scenario packs. Concentration is not evaluated in a single world; it is evaluated across worlds where diversification can fail.

3.2.11 Latency-aware constraints: enforcing limits under delayed measurement

A defining feature of real trading systems is latency and asynchrony. Signals arrive with delays, orders execute over time, risk systems update on cadence, and portfolios are measured with lags. These lags create a governance problem: a controller can violate constraints in real time even if it appears compliant in a synchronous simulation. Institutions therefore build latency-aware buffers and enforce constraints conservatively.

In a surrogate lab, we cannot replicate full microstructure latency, but we can encode its essential governance implications. First, we can model delayed observation: the controller sees s_t with delay, or certain features are lagged. Second, we can model execution lag: actions affect holdings over multiple steps rather than instantly. Third, we can model delayed constraint measurement: constraints are evaluated on a lagged portfolio snapshot, creating risk that the controller overshoots.

Latency-aware constraints can be implemented by tightening limits to include a buffer for uncertainty. For example, if leverage cap is L, enforce an internal cap $L_{\text{int}} < L$ to account for measurement delay. Similarly, enforce a turnover cap below the operational maximum. The buffer values are governance decisions, and they must be logged. The controller then learns under these internal caps, ensuring that real-time operations remain safe.

Latency-aware constraints can also be encoded through state-dependent tightening. When volatility rises, latency risk increases because prices move faster during execution. Therefore, buffers should increase in high volatility. The controller can implement this by tightening effective leverage and turnover caps when volatility is high. This is consistent with institutional practice: “be more conservative when the world is moving fast.”

The key point is that latency-aware constraints transform the controller from a theoretical optimizer into an operational system. If a controller is trained without acknowledging latency, it can learn behavior that assumes instantaneous rebalancing and perfect monitoring. Such behavior is not deployable. By encoding latency-aware buffers, the surrogate controller learns to operate with margin and to avoid aggressive boundary behavior.

3.2.12 Synthesis: constraint signals as the institutional contract

This section defines what institutions actually control and how those controls become constraint signals in a CMDP. Drawdown is encoded as a procedural trigger and a budget. Turnover and participation define operational feasibility. Tail risk is constrained through CVaR and stress-conditional budgets. Concentration limits prevent hidden one-factor exposures and are stress-tested under correlation compression. Latency-aware buffers ensure that constraints remain safe under delayed measurement and execution.

Together, these signals form the institutional contract: the controller is allowed to optimize performance only within these mandates, with buffers and state-dependent tightening. The next section will show how we enforce this contract during learning: primal–dual training and governance signal dynamics. Constraint signals define what must be respected; primal–dual mechanics define how the controller learns to respect them.

3.3 Primal–Dual Training as Governance Mechanics

3.3.1 Lagrangian relaxation and multipliers as shadow prices

Once constraints are explicit, the next institutional problem is enforcement during learning. We need a training mechanism that can improve performance while respecting multiple constraints, without relying on ad hoc penalty tuning. Primal–dual methods provide a principled answer by transforming a constrained optimization problem into a saddle-point problem: a policy (the primal object) is optimized to maximize reward, while a set of multipliers (the dual object) is optimized to penalize constraint violations until budgets are satisfied.

At a high level, consider a CMDP with reward $R(\pi)$ and constraint costs $C_i(\pi)$ for $i = 1, \dots, m$, with budgets d_i. The constrained problem is:

$$\max_{\pi} \; R(\pi) \quad \text{s.t.} \quad C_i(\pi) \le d_i \quad \forall i.$$

The Lagrangian relaxes this by introducing multipliers $\lambda_i \ge 0$:

$$\mathcal{L}(\pi, \lambda) = R(\pi) - \sum_{i=1}^{m} \lambda_i \left(C_i(\pi) - d_i\right).$$

The primal–dual objective is then:

$$\max_{\pi} \; \min_{\lambda \ge 0} \; \mathcal{L}(\pi, \lambda),$$

or equivalently $\max_\pi$ and $\max_{\lambda \ge 0}$ depending on sign conventions; the key is that λ is adjusted to enforce constraints. The institutional meaning of λ is shadow price: it measures how much reward the system is willing to sacrifice to reduce constraint cost. When λ_i is high, the institution is "charging" the policy heavily for consuming constraint budget i. When λ_i is near zero, the constraint is slack.

This shadow-price interpretation is the central governance advantage of primal–dual training. Penalty weights in a proxy objective are arbitrary. Multipliers are not. They are learned signals that respond to observed violations. If turnover exceeds its budget, the turnover multiplier rises, increasing pressure to reduce turnover. If drawdown risk remains within budget, the drawdown multiplier can remain low. This adaptive process is what turns constraints from static thresholds into dynamic governance mechanisms.

However, we must be explicit: the Lagrangian relaxation is not automatically safe. If λ is updated aggressively, the system can oscillate. If the policy class cannot satisfy the constraints, multipliers can diverge. Therefore, Chapter 3 treats primal–dual mechanics as both an optimization algorithm and a governance system that must be stabilized and audited. The mathematical mechanism becomes an institutional instrument only if it is stable, bounded, and interpretable.

3.3.2 Multiplier dynamics: stability, step sizes, and clipping

In practice, primal–dual training is an iterative process. At each iteration, the policy is updated to improve the Lagrangian objective given current multipliers, and multipliers are updated based on observed constraint costs. The core governance question is: how do we update multipliers in a way that enforces constraints without destabilizing policy behavior?

A standard update is:

$$\lambda_i \leftarrow \left[\lambda_i + \eta_\lambda \left(\widehat{C_i} - d_i\right)\right]_+,$$

where $\widehat{C_i}$ is an estimate of cost under the current policy, η_λ is a dual step size, and $[\cdot]_+$ projects onto nonnegative values. This resembles a feedback controller: if cost exceeds budget, increase λ; if cost is below budget, decrease λ (or let it decay). The dual step size determines responsiveness. High responsiveness can enforce constraints quickly but can induce oscillation. Low responsiveness can be stable but slow, allowing extended periods of violation.

Institutions prefer stability over speed. A trading controller that oscillates between extreme risk-on and risk-off behavior is not acceptable, even if it "converges" eventually. Therefore, dual updates must be stabilized. There are several institutional-grade stabilizers.

First, smoothing of cost estimates. Constraint costs are noisy, especially tail measures. Using raw one-trajectory estimates can cause multipliers to react to noise. Instead, one can use moving averages or batch estimates across multiple seeds or scenarios. In a governed synthetic lab, it is feasible to compute costs across a scenario pack and update multipliers based on aggregated evidence, reducing noise-induced oscillation.

Second, clipping of multipliers. Multipliers can diverge if constraints are infeasible or if noise produces repeated apparent violations. Diverging multipliers can collapse learning by overwhelming reward. Therefore, institutional pipelines clip λ to a maximum $\lambda_{\max}$ and treat reaching $\lambda_{\max}$ as a failure signal: either the constraints are infeasible, or the policy class cannot satisfy them. This turns divergence into an auditable event rather than a silent numerical pathology.

Third, step-size schedules. Dual step sizes can decay over time, reducing oscillations as the system approaches feasibility. Alternatively, one can use separate time scales: update the policy more frequently than multipliers, or vice versa. A common stability approach is to update multipliers slowly relative to policy updates, allowing the policy to adapt before the governance signal changes again. The exact choice is a governance decision because it affects behavioral stability.

Fourth, constraint slack targets. Instead of targeting $C_i \leq d_i$ exactly, institutions often target $C_i \leq d_i - \epsilon_i$ to create buffer. This can be implemented by setting budgets with margin. The multiplier dynamics then enforce not only compliance but compliance with buffer. This helps reduce boundary time and makes the learned controller more deployable.

Finally, dual regularization. One can penalize large multipliers or changes in multipliers to encourage smooth governance. This is not about making constraints weaker; it is about making enforcement smoother. If the multiplier changes too abruptly, the policy may react violently, producing instability. A regularized dual update helps produce stable governance signals.

In Chapter 3, these stabilizers are not optional enhancements. They are part of institutional production readiness. A primal–dual algorithm that is unstable is not a governance mechanism; it is a risk generator.

3.3.3 Interpreting multipliers as governance signals (not just math)

The multipliers λ_i are not merely algorithmic parameters; they are outputs that can be monitored, audited, and escalated. This is a key institutional upgrade. Under penalty objectives, there is no meaningful way to interpret a fixed penalty weight as a signal. Under primal–dual training, the multipliers reflect constraint tension: how hard the system must push to remain compliant.

Each multiplier corresponds to an institutional domain. A turnover multiplier corresponds to operational feasibility pressure. A drawdown multiplier corresponds to capital preservation pressure. A tail-risk multiplier corresponds to crash vulnerability pressure. A concentration multiplier corresponds to diversification pressure. When these multipliers rise, the system is telling you that without strong governance pressure, the policy would violate the constraint.

This interpretation supports several institutional uses.

First, diagnostics. If a policy achieves high reward but requires high multipliers to satisfy constraints, it is likely fragile. It is living near a cliff. In contrast, a policy that satisfies constraints with low multipliers is more naturally aligned. Multipliers therefore become part of acceptance criteria: not only "are constraints satisfied," but "how much governance pressure is required to satisfy them."

Second, regime attribution. Because multipliers can be updated based on regime-conditional costs, one can observe which regimes trigger which constraint tensions. For example, tail multipliers may spike in crisis regimes, turnover multipliers may spike in transition regimes, concentration multipliers may spike under correlation compression. These patterns help supervisors understand when the controller is at risk and what to monitor.

Third, operational monitoring in production candidates. Even if Chapter 3 is synthetic, the controller object can be designed to output real-time estimates of constraint costs and internal multipliers. In production, one might not update multipliers autonomously, but one can monitor "constraint pressure indices" that play a similar role. These indices can trigger governance overlays: if turnover pressure rises, reduce trading; if tail pressure rises, de-risk; if concentration pressure rises, diversify. This turns primal–dual outputs into operational safety features.

Fourth, change management. If a policy revision causes multipliers to rise, it indicates that the revision increased constraint tension. This can be treated as a regression signal even if performance improved. Institutions often reject changes that increase tension because tension correlates with fragility.

Thus, multipliers serve as a bridge between mathematical optimization and institutional supervision. They convert abstract constraints into measurable pressure signals. This is why Chapter 3 emphasizes primal–dual training: it produces not only a policy but also governance telemetry.

3.3.4 Failure modes: oscillation, constraint chasing, and instability

Primal–dual training can fail, and its failure modes are institutionally important because they resemble real governance failures: overreaction, delayed response, and oscillation. Chapter 3 explicitly documents these failure modes and incorporates controls to detect and mitigate them.

The first failure mode is oscillation. If multipliers update too aggressively, the system can oscillate between satisfying constraints and violating them. For example, a high turnover multiplier may push the policy to trade less, which may reduce reward, which may push the policy to take more risk elsewhere, which may increase drawdown, which may increase drawdown multiplier, and so on. The result can be a "bang-bang" controller that alternates between extreme behaviors. This is unacceptable institutionally because it can cause operational stress and unpredictable exposures.

The second failure mode is constraint chasing. In constraint chasing, the policy learns to exploit the lag in constraint enforcement. For example, if the constraint is measured as a rolling average turnover, the policy may trade aggressively early and then trade less later to keep the average within budget. This can still be unacceptable if the aggressive period coincides with stress. Therefore, constraint definitions must be chosen carefully, and acceptance must include pointwise constraints and boundary time, not only averages.

The third failure mode is infeasibility masquerading as learning. If constraints are too tight, the multipliers can rise to maximum and the policy can collapse to trivial behavior (e.g., go flat). This may satisfy constraints but produces no meaningful operation. This is not a success; it indicates that the contract is infeasible for nontrivial performance. The proper response is to revise constraints, environment, or policy class.

The fourth failure mode is numerical instability. In practice, constrained learning involves noisy estimates of costs and gradients. Numerical issues can cause divergence, especially when tail measures are involved. Institutional-grade pipelines detect these issues by monitoring multiplier trajectories, constraint margins, and training stability metrics. They also enforce safe stops: if multipliers saturate, if breaches spike, or if behavior becomes unstable, the run is terminated and recorded.

These failure modes underscore that primal–dual training is not “solve once and done.” It is a controlled governance mechanism that requires monitoring and design. Chapter 3 treats these issues as central because they determine whether constrained learning is stable enough to be supervision-ready.

3.3.5 Auditable training logs: what must be recorded every iteration

A primal–dual training process is only institutional if it is auditable. Auditability here means that a reviewer can reconstruct what happened, why multipliers changed, and how the policy responded. Therefore, Chapter 3 defines a minimum training log standard that must be exported every iteration (or at a defined cadence, such as per epoch).

At minimum, the log must record: the run$_i d, UTCtimestamp, configurationhash, iterationindex, policyparameter$ each budget d_i, each multiplier λ_i, and derived quantities such as constraint margins $(d_i - \widehat{C_i})$. It must also record stability diagnostics: step sizes used, clipping events, and whether any safe-stop criteria were triggered.

The log should also record scenario context if training uses scenario packs: which scenarios were sampled, what regime conditions were present, and how costs decomposed by scenario category. This supports attribution: if a multiplier spike occurs, one can see whether it was driven by a crisis scenario or by execution inflation.

Importantly, these logs are not just for debugging. They are part of governance evidence. They allow a committee to see that constraints were enforced during learning, not merely evaluated after. They also allow independent evaluation: another reviewer can rerun training with the same seeds and confirm the log trajectories match within tolerances. This is the institutional version of reproducibility.

Finally, the logs should be exported as structured JSON lines (or equivalent) to enable automated checks. Governance should not rely on manual inspection. Automated validators can check: no missing fields, consistent hashes, monotone iteration indices, multipliers within bounds, and constraint costs within budgets after convergence. This is how constrained learning becomes production-grade.

3.3.6 Synthesis: primal–dual training as a supervised control layer

This section establishes primal–dual training not merely as a mathematical method but as a supervised governance layer. The Lagrangian multipliers become shadow prices that quantify constraint tension. Their dynamics must be stabilized with smoothing, clipping, step-size control, and buffer targets. The multipliers become interpretable governance signals that can be used for diagnostics and monitoring. The training process has recognizable failure modes that must be detected and managed. And the entire process must produce auditable logs that support reproducibility and committee review.

With primal–dual governance mechanics defined, the next section extends the learning loop to robustness: training under scenario packs and domain randomization so that constraint compliance is not a fragile property of a single synthetic world. Constrained learning enforces mandates; robust constrained learning enforces mandates across uncertainty.

3.4 Robust Constrained Learning Under Scenario Packs

3.4.1 Domain randomization: training across environments, not one world

A controller that is constraint-respecting in one synthetic world can still be institutionally unsafe. The central reason is distribution shift: the environment will change, regimes will occur in different sequences, liquidity will deteriorate in different patterns, and tail events will arrive at different times. Chapter 3 therefore treats robustness not as an after-the-fact evaluation step but as a training design principle. The institutional goal is not "a policy that works in the sampled world," but "a policy that remains within governance boundaries across a family of plausible worlds."

Domain randomization is the primary mechanism for this. Instead of training under a single environment parameterization, we define an environment family indexed by parameters θ that govern regime prevalence, switching speed, volatility-of-volatility, correlation dynamics, and liquidity conditions. Each training episode samples θ from a bounded distribution representing uncertainty. The policy then learns a behavior that performs and remains compliant across that distribution. In effect, the policy is trained to be robust to environment variability.

Institutional governance requires that the domain randomization distribution itself be governed. It must be justified and bounded. Parameters cannot be chosen opportunistically to produce easy learning. They must represent plausible stress and plausible model error. The distribution must be versioned, logged, and treated as part of the contract. If the distribution changes, the controller must be retrained and reapproved. This is the same discipline applied to risk models in institutions: model assumptions are versioned and reviewed.

Domain randomization also interacts with constraints. Constraints can be easier to satisfy in some environments and harder in others. If the environment family includes worlds where liquidity collapses and volatility spikes, turnover constraints become tighter and tail risk becomes more severe. A robust constrained learner must therefore internalize a state-dependent conservatism: trade less when liquidity is poor, reduce exposure when volatility rises, avoid concentration when correlation compresses. This is precisely the kind of behavior institutions want. It is not an overlay added after training; it is behavior learned because training repeatedly exposes the controller to conditions where failing to behave conservatively causes constraint violations.

Finally, domain randomization provides a form of "anti-overfitting." When a policy is trained on many worlds, it is harder to exploit idiosyncrasies of one world. Proxy hacking becomes less likely because the loophole must work across worlds. This does not eliminate exploitation, but it raises the bar and makes constraints more likely to be satisfied robustly.

3.4.2 Stress-aware sampling: forcing tail regimes into the learning signal

Even with domain randomization, tail regimes can remain underrepresented if they are rare. In real markets, crises are rare but decisive. A controller that fails in crises is unacceptable even if it performs well most of the time. Therefore, institutional robustness requires stress-aware sampling: deliberately increasing the frequency or weight of tail regimes in training so that the learning signal includes them.

Stress-aware sampling can be implemented in several ways. One method is to oversample episodes whose parameters θ correspond to crisis-like conditions: high volatility, high correlation, low liquidity, high jump intensity. Another method is to include a fixed fraction of "stress episodes" in each training batch. A third method is to reweight the learning objective to emphasize performance and constraint compliance in stress states, for example by applying state-dependent weights or by using risk-sensitive objectives such as CVaR across episodes.

The institutional purpose is clear: do not let the controller learn that crises "do not matter." If crises appear only occasionally, the optimizer may allocate capacity to improving performance in normal conditions and accept rare catastrophic failures. That is not institutionally acceptable. Stress-aware sampling makes crises a regular part of the training experience, forcing the controller to learn conservative behavior in those states.

Stress-aware sampling must be governed to avoid unrealistic pessimism. If stress frequency is set too high, the controller may become overly conservative and underperform. Institutions do not want paranoia; they want controlled risk. Therefore, stress sampling rates must reflect a governance choice: they represent the institution's risk appetite and its view of tail uncertainty. In a synthetic lab, the choice is explicit and can be documented.

A key advantage of stress-aware sampling is that it improves constraint feasibility. Many constraints are most likely to be violated in stress. If the policy learns during training how to behave in stress, constraint violations become less likely in evaluation. This reduces reliance on stop-outs and emergency overlays, making behavior smoother and more predictable.

3.4.3 Worst-case and CVaR-style objectives under constraints

Robust constrained learning can be formulated not only through environment randomization and sampling, but also through the structure of the objective. Standard expected reward is not an institutional objective because it can hide tail disasters. Institutions often prefer objectives that are conservative: maximize a lower quantile of performance, or maximize expected reward subject to CVaR constraints, or minimize worst-case loss. Chapter 3 integrates these ideas within the CMDP framework.

One approach is to define reward as a risk-sensitive utility, such as the negative of a coherent risk measure. CVaR is a common choice. For example, define the episodic return G and optimize $\mathrm{CVaR}_\alpha(G)$ rather than $\mathbb{E}[G]$. This pushes the policy to improve performance in the worst α fraction of episodes. However, pure CVaR optimization can be too conservative and can interact unpredictably with constraints. Therefore, institutions often use CVaR as a constraint rather than as the objective: require that $\mathrm{CVaR}_\alpha(\text{loss}) \leq \tau$. This is directly compatible with the CMDP framework by defining a tail cost signal.

Worst-case objectives are even more conservative. For scenario packs, one can define a robust value as the minimum performance across scenarios in the pack and attempt to maximize that minimum subject to constraints. This is essentially a min-max formulation: the environment chooses a scenario to minimize reward, and the policy chooses actions to maximize reward under that adversary. Full min-max optimization can be complex, but even approximate worst-case evaluation can guide training: update the policy based on the worst-performing scenarios or on a subset of adverse scenarios.

The institutional interpretation is that robust objectives reduce sensitivity to model error. If the controller is optimized for average conditions, it may fail when the world deviates. If the controller is optimized for worst-case or tail conditions, it is more likely to remain safe. The trade-off is performance: robust objectives often reduce average returns. Institutions accept this trade-off when the alternative is unacceptable tail risk.

Within primal–dual training, CVaR and worst-case ideas can be integrated by defining cost signals that reflect tail outcomes and by updating multipliers based on tail estimates across scenarios. For example, one can estimate tail costs across a batch of episodes and use the worst tail costs to update multipliers, making enforcement sensitive to tails rather than averages.

3.4.4 Robustness to model error: bounded perturbations during training

An institution cannot assume its model of execution, regimes, and dynamics is correct. Even a synthetic lab must treat its environment as a proxy for reality, and therefore must model uncertainty about that proxy. Robust constrained learning addresses this by injecting bounded perturbations during training that represent model error. The goal is to train controllers that remain compliant even when the environment differs moderately from the assumed model.

Perturbations can target several dimensions. Dynamics perturbations: change regime switching probabilities, drift and volatility parameters, and jump intensities within bounds. Observation perturbations: increase noise, distort signals, introduce lag. Execution perturbations: inflate spread and impact parameters, increase convexity exponent, reduce liquidity depth, introduce occasional execution shocks. Correlation perturbations: compress correlations in crises, reduce dispersion, increase cross-sectional synchronization. Each perturbation corresponds to a plausible source of model risk.

The perturbation budget must be explicit and governed. It defines how pessimistic training is. Too small a budget yields fragile controllers. Too large a budget yields overly conservative controllers. The budget should be set based on institutional risk appetite and on the expected model error in the deployment context. In the synthetic lab, the budget is a scenario design choice, but in institutional terms, it represents how much error the institution is willing to tolerate.

Bounded perturbations also serve as exploitation defenses. If a policy exploits a narrow modeling assumption—for example, that impact costs are mild—then perturbing impact costs will break that exploit. Training under perturbations forces the policy to learn behaviors that remain feasible when assumptions shift. This reduces the chance that a policy is “good” only because the model is optimistic.

In a robust constrained pipeline, perturbations are applied during training, not only evaluation. This is the key upgrade. It forces the policy to internalize uncertainty and to behave conservatively where needed. It also makes constraint compliance more reliable across unseen conditions.

3.4.5 When robustness conflicts with reward: institutional trade-off rules

Robustness is not free. The more robust we make training—more stress sampling, more pessimistic perturbations, more conservative objectives—the more likely we are to reduce performance. Institutions do not want robustness for its own sake. They want a controlled trade-off: enough robustness to make the system safe and predictable, but not so much that the system becomes trivial or unusable.

Therefore, Chapter 3 introduces institutional trade-off rules. These rules are part of the stage-gate contract: define minimum acceptable performance relative to baselines, define maximum acceptable risk and constraint tension, and define acceptable conservatism. For example, an institution might require that the controller achieve at least a certain fraction of baseline return while keeping CVaR below a budget and turnover below a budget. If robustness training reduces performance below this minimum, the controller is not acceptable even if it is safe, because the institution could achieve similar safety with simpler baselines.

Trade-off rules also apply to constraint margins. Excessively conservative controllers that stay far from constraints may be acceptable but may be inefficient. Institutions often have a desired "operating region" where constraints are respected with buffer but not wasted. For example, maintain leverage below cap but not always at zero. Maintain turnover below cap but not always at minimal levels. The operating region is a governance choice reflecting risk appetite.

Trade-off rules must be auditable. The notebook must report how robustness settings affected performance and constraint tension. If adding stress sampling reduced tail risk but also reduced return, the institution must decide whether the trade is acceptable. This decision should be based on evidence, not on narrative. Therefore, Chapter 3 requires that robustness parameters be swept and that the impact on metrics be recorded in the artifact bundle.

Finally, trade-off rules define escalation. If robustness training reveals that achieving acceptable performance requires operating too close to constraints or requires high multipliers, the institution may decide to stop. This is a valid outcome: it indicates that the mechanism is not robust enough for the given mandate. Robust constrained learning thus becomes a truth-telling process: it reveals whether a strategy mechanism can exist within institutional constraints under uncertainty.

3.4.6 Synthesis: robust constrained learning as institutional realism

This section integrates robustness into constrained learning. Domain randomization ensures the controller is trained across a family of plausible worlds, not one path. Stress-aware sampling forces tail regimes into the learning signal so that crises shape behavior. CVaR and worst-case ideas provide conservative training objectives and constraints that align with institutional risk management. Bounded perturbations represent model error and reduce exploitability. Trade-off rules define how robustness and performance are balanced and how acceptance decisions are made.

The result is an institutional-grade training posture: the controller is trained under mandates across uncertainty, and acceptance is defined by reproducible compliance and acceptable performance. The next section will address an equally important design dimension: policy classes. Robust constrained learning can fail if the policy class is too weak to satisfy constraints, or it can become ungovernable if the policy class is too complex. Choosing a policy class under supervision requirements is therefore a central institutional decision.

3.5 Policy Classes for Constrained Surrogate Controllers

3.5.1 Interpretable linear controllers with constraint-aware scaling

In an institutional setting, "policy class" is not a purely technical choice. It is a governance choice. The policy class determines how the controller can behave, how it can fail, how it can be audited, and how it can be explained. Chapter 3 is explicitly about constraint-respecting control, so the policy class must support bounded actions, stable behavior, and interpretability of constraint interactions. The most defensible starting point is the interpretable linear controller, augmented with constraint-aware scaling.

A linear controller can be written as $a_t = \Pi_{\mathcal{A}(s_t)}(K\phi(s_t) + b)$, where $\phi(s_t)$ is a bounded feature map of the operational state, K is a parameter matrix, b is a bias, and $\Pi_{\mathcal{A}(s_t)}$ projects the proposed action onto the feasible action set $\mathcal{A}(s_t)$, which may be state-dependent due to liquidity or risk tightening. This form is powerful because it separates three components clearly: representation, decision rule, and feasibility enforcement. Each is auditable.

Constraint-aware scaling means that the action is not only clipped by hard limits, but also scaled based on constraint margins and governance signals. For example, if the drawdown margin is tight or if the turnover budget is close to binding, the controller can scale down exposure or scale down trading intensity. This can be encoded explicitly in the feature map: include constraint margins, volatility estimates, liquidity multipliers, and even dual variables as features. The linear controller then learns how to respond smoothly to these governance signals rather than relying solely on discrete stop-outs.

The institutional value of this class is that behavior is explainable. If exposure decreases when crisis belief rises, that is interpretable. If turnover decreases when liquidity deteriorates, that is interpretable. The parameters K can be inspected and constrained. For example, one can enforce sign constraints or monotonicity constraints on certain coefficients to guarantee conservative behavior (e.g., higher drawdown implies lower exposure). This is an important point: interpretable controllers can be governed not only by constraints but also by structural parameter restrictions that prevent unsafe couplings.

Linear controllers also tend to be stable and easier to train under primal–dual dynamics. Because their capacity is limited, they are less likely to discover complex proxy loopholes. This can be an advantage in early institutional pipelines: it reduces the risk that optimization produces clever but fragile behaviors. If the linear class can meet acceptance criteria, the institution may prefer it over more complex classes, even if performance is slightly lower.

Finally, linear controllers provide a baseline policy class for evaluating whether constraints are feasible at all. If even a simple linear controller cannot satisfy constraints while producing nontrivial behavior, the contract may be infeasible or the environment too adversarial. Thus, interpretable linear controllers are both a production candidate class and a diagnostic instrument.

3.5.2 Nonlinear controllers: capacity, risk, and explainability costs

Nonlinear controllers—such as multilayer perceptrons or other function approximators—expand capacity. They can represent complex state-action mappings, capture nonlinear interactions among features, and adapt behavior across regimes in ways linear controllers cannot. This can improve performance and robustness. But in institutional settings, capacity is not free. It increases governance cost: explainability declines, failure modes become more complex, and oversight becomes harder.

The central risk of nonlinear controllers under optimization pressure is that they can discover subtle exploit patterns that are difficult to detect. Even in a constrained setting, a nonlinear policy can "thread the needle" by finding behaviors that satisfy constraints in expectation but concentrate risk in rare corners of the state space. It can also exploit imperfections in state representation: if certain constraint-relevant variables are approximated, the nonlinear policy can learn to manipulate those approximations. This is analogous to reward hacking, but in a constrained setting it becomes constraint gaming: satisfying measured constraints while violating the underlying intent.

Therefore, if nonlinear controllers are used, they must be governed by stronger controls. First, their inputs must be bounded and well-defined. Feature maps must be normalized and clipped to prevent extreme extrapolation. Second, the action projection must be rigorous and state-dependent, ensuring hard constraints are enforced by construction. Third, training must include strong robustness randomization and stress sampling, so that the nonlinear policy cannot exploit a narrow region of the environment family. Fourth, acceptance criteria must include stronger diagnostics: boundary time, multiplier tension, tail concentration, and generalization gaps.

Explainability costs must also be acknowledged. Institutions can sometimes accept nonlinear policies if they are accompanied by credible explanations: sensitivity analyses, monotonicity constraints, or surrogate interpretability tools. But these tools are often weaker than direct interpretability of linear policies. Therefore, the governance standard for nonlinear controllers must include explicit documentation of why the additional complexity is justified: what performance and robustness gains were obtained, and what controls mitigate the added risk.

A key institutional concept is "complexity budget." The policy class is allowed to become more complex only if evidence shows that simpler classes cannot meet mandates without unacceptable performance loss. This creates a disciplined escalation: start simple, escalate only if necessary. Chapter 3 aligns with this discipline.

Finally, nonlinear controllers can be constrained structurally. One can design architectures that embed safety: for example, a nonlinear network that outputs a risk appetite scalar which is then combined with a conservative baseline controller, rather than outputting raw actions. This hybrid approach can preserve some interpretability and reduce risk of extreme behavior. Thus, "nonlinear" need not mean "unbounded."

3.5.3 Stochastic policies: uncertainty-aware exploration under supervision

Stochastic policies introduce randomness in action selection. In reinforcement learning, stochasticity supports exploration and can improve robustness by avoiding deterministic brittle choices. In trading, stochasticity can represent uncertainty-aware behavior: when the system is unsure, it may randomize slightly among conservative actions. However, institutions are cautious about stochasticity because it can complicate supervision: two runs may behave differently, and auditors may struggle to reproduce behavior in detail.

In a governed surrogate pipeline, stochastic policies can still be acceptable if stochasticity is controlled. First, randomness must be seeded and logged so that behavior is reproducible. Second, the action distribution must be bounded and constrained: randomness should not allow actions outside the feasible set. In practice, one can sample a latent action and then project it onto feasibility constraints, ensuring hard compliance. Third, stochasticity must be limited in magnitude, especially near boundaries. A policy that is stochastic near leverage caps or turnover caps is dangerous because randomness can produce breaches in practice, especially under latency.

Stochastic policies can be institutionally valuable when they represent uncertainty. For example, if regime belief is diffuse and signals are noisy, the policy may choose a lower exposure with some small randomized variation among a conservative set. This can reduce sensitivity to estimation error. Stochasticity can also help avoid pathological loops where deterministic policies repeatedly switch in response to small changes in ranking signals, producing churn. A carefully designed stochastic policy can smooth such behavior.

However, institutional acceptance usually requires that stochasticity be "explainable stochasticity," not arbitrary randomness. One should be able to say: "the policy randomizes among conservative actions because uncertainty is high," and one should be able to quantify the distribution. Therefore, the policy should output not only an action but a distribution parameterization and uncertainty measures. These can be logged and reviewed.

In primal–dual constrained learning, stochastic policies introduce additional complexity because cost estimates become noisy. This increases the risk of multiplier oscillation. Therefore, if stochastic policies are used, the training protocol must use larger batches, more smoothing, and possibly slower dual updates to maintain stability. This is an institutional trade-off: stochasticity can improve robustness but increases governance burden.

Finally, institutions often prefer to deploy deterministic policies even if training uses stochastic exploration. One common pipeline is: train with stochasticity, then distill or select a deterministic policy that approximates the learned behavior, and validate it under the same acceptance suite. This yields reproducible production behavior while still benefiting from exploration during training. Chapter 3 is compatible with this pipeline because it treats the controller as a versioned artifact with acceptance evidence.

3.5.4 Hybrid controllers: rule-based overlays + learned core

Hybrid controllers are often the most institutionally defensible policy class in early deployment contexts. The idea is to combine a learned core with rule-based governance overlays that enforce constraints and encode known safe behaviors. This reflects institutional reality: most real trading systems are layered. There is an alpha model, an execution model, and a risk overlay. Autonomy is constrained by governance. Hybrid controllers formalize this layering.

A common hybrid pattern is: start with a conservative baseline controller (e.g., volatility targeting with drawdown overlays), and allow a learned component to modulate certain parameters or to choose among a small set of safe modes. For example, the learned core may output a risk appetite scalar $\rho_t \in [0, 1]$ that scales exposure within the baseline's limits. Or it may choose among discrete modes: normal, cautious, crisis, each with different caps. Another pattern is that the learned component outputs a desired portfolio tilt, but the overlay projects it onto feasible weights with strict constraints and additional conservative buffers.

The institutional advantage is that the overlay provides hard safety. Even if the learned component behaves unexpectedly, the overlay prevents catastrophic actions. The learned component can improve performance by adjusting within safe ranges, but it cannot override mandates. This reduces the risk of constraint gaming. It also improves interpretability: the baseline and overlay provide a reference mechanism, and the learned component's role is bounded.

Hybrid controllers also align well with primal–dual governance. The multipliers can be interpreted at two levels: they can influence the learned component to reduce pressure, and they can also inform overlay tightening. For example, if turnover multiplier is rising, the overlay can tighten turnover caps, and the learned component can learn to propose smoother actions. This creates a coordinated governance system.

However, hybrid controllers must be designed carefully to avoid false security. If the overlay is too permissive or if it can be gamed, the learned component may still exploit. Therefore, the overlay itself must be audited and stress-tested. It must enforce buffers and must be latency-aware. The overlay is part of the controller object and must be versioned.

Hybrid controllers also facilitate committee review. A committee can often accept a learned modulator of a conservative baseline more readily than a fully learned policy. This is because the baseline's behavior is familiar and the learned component is constrained. In many institutions, this is the pragmatic path: deploy learned components gradually, under strong overlays, and expand autonomy only with evidence.

3.5.5 Choosing a class under committee review requirements

The final decision is not "which class is best in theory," but "which class can be accepted under supervision requirements." Committee review requirements include interpretability, reproducibility, stability, and failure containment. The policy class must be chosen to satisfy these requirements.

A disciplined selection process begins with the simplest viable class. If an interpretable linear controller meets constraints with buffer, passes robustness evaluation, and outperforms baselines acceptably, it is often the preferred choice. Complexity is then unnecessary and adds risk. If the linear class cannot meet performance thresholds or cannot satisfy constraints without going trivial, then escalation is justified.

Escalation should be evidence-driven. The evidence comes from Chapter 2 and Chapter 3 experiments: which failure modes persist under the simpler class, and what performance gap remains. If the gap is large and robust, nonlinear or hybrid classes may be justified. But acceptance criteria must become stricter as complexity increases. This is a governance principle: higher capability requires higher control.

Committee review also prefers bounded autonomy. Hybrid controllers often provide the best trade-off: they allow learning to improve performance but keep mandates enforced by overlay. Stochastic policies are typically acceptable in training but less acceptable in deployment unless distilled or constrained tightly.

Ultimately, the policy class choice must be documented as part of the audit bundle. The documentation should explain: what classes were tried, what acceptance evidence was obtained, what trade-offs were observed, and why the chosen class is the best institutional compromise. This transforms model selection from a research preference into a governance decision.

3.5.6 Synthesis: policy class selection as governance, not taste

This section reframes policy class selection as an institutional design choice. Interpretable linear controllers provide stability and auditability and often suffice. Nonlinear controllers provide capacity but increase governance risk and require stronger controls. Stochastic policies can improve robustness but complicate supervision and require reproducibility controls. Hybrid controllers align with institutional layering and provide bounded autonomy. The final selection must satisfy committee review requirements and must be evidence-driven.

With policy classes defined, the next section moves to acceptance: evaluation and stage gates for constrained controllers. Constraint-respecting learning is only institutional if acceptance is reproducible, distributional, and comparative to baselines. The controller must not only learn; it must be approved.

3.6 Evaluation and Acceptance: Reproducible Stage Gates for Constrained Controllers

3.6.1 Acceptance criteria: constraint compliance distribution, not point estimates

Institutional acceptance is not a celebration of a single good run. It is a claim about a distribution of outcomes under uncertainty. A constrained controller is accepted only if it demonstrates constraint compliance and acceptable performance *repeatedly* across independent environment draws, across stress regimes, and across realistic execution degradations. This requirement is not bureaucratic. It is the operational logic of risk control: the institution is committing capital to a system whose behavior will unfold in many possible future paths, not in the one path used to train the model.

Therefore, acceptance criteria must be distributional. Instead of asking "did the policy breach a constraint in the evaluation run," we ask "what is the breach probability across held-out seeds and scenario packs," and "what is the distribution of near-breaches and boundary living." Similarly, instead of asking "what is the average turnover," we ask "what is the upper tail of turnover across regimes," because operational failure is driven by peaks, not means. This is the institutional shift from point estimates to compliance distributions.

A minimum acceptance standard typically includes: (i) near-zero breach probability for hard constraints, (ii) bounded tail metrics for losses (e.g., CVaR below a budget), (iii) bounded drawdown across scenarios, and (iv) stable feasibility under execution cost inflation. But the key is how these are measured. For each constraint, we define a distribution across evaluation replications: multiple deterministic seeds, multiple scenario configurations, and multiple perturbation draws. For each replication, we compute constraint outcomes: breach indicator, maximum exceedance, time above threshold, and boundary time. We then summarize the distribution with conservative statistics: worst-case, high quantiles, and exceedance rates.

Institutions often define a "confidence envelope." For example, accept only if the 95th percentile of maximum drawdown is below the drawdown limit with buffer, and the 99th percentile of turnover peak is below turnover cap with buffer. The exact percentiles reflect risk appetite. In a synthetic lab, the percentiles are computed from the evaluation distribution; in a production pipeline, they would be updated with live monitoring. The acceptance standard should also specify sample size: how many seeds and scenarios are required to claim a percentile. This prevents gaming by using too few replications.

This distributional approach also reduces Goodhart risk. A policy can be tuned to pass a single evaluation path. It is harder to tune to pass a large distribution unless its behavior is genuinely robust. Thus, distributional acceptance is both a governance safeguard and a scientific discipline.

Finally, distributional acceptance must be tied to explicit tolerances. No simulation is perfect, and some rare numerical anomalies can occur. Institutions sometimes allow "near-zero" breach rates rather than absolute zero, but only under strict definitions: what counts as a breach, how large it was, and whether it occurred in realistic scenarios. These tolerances must be predeclared and logged. Otherwise, acceptance becomes discretionary and vulnerable to narrative drift.

3.6.2 Generalization across seeds and structural breaks

Generalization is the center of credibility. A constrained controller that performs well only in the training seed is not a mechanism; it is an artifact. Chapter 3 therefore treats generalization not as an abstract ML goal but as a governance necessity. Generalization is assessed along two axes: (i) stochastic variability (different seeds, different regime paths), and (ii) structural breaks (changes in environment structure that represent model error or regime shifts).

Cross-seed generalization is the baseline test. Evaluation must include held-out seeds that were not used in training or tuning decisions. These seeds generate different regime sequences, different return realizations, and different execution conditions. The controller must remain compliant across them. Performance should be stable in a distributional sense: not necessarily identical returns, but similar risk profiles, similar constraint margins, and similar diagnostic signatures. Large variance in outcomes, especially in tail metrics and boundary time, is a warning that the controller is fragile.

Structural breaks are more demanding. They represent changes in the environment family: shifts in regime transition probabilities, changes in correlation dynamics, sudden liquidity regime changes, and changes in volatility-of-volatility. In real markets, structural breaks are the norm over long horizons. A controller that assumes stationary dynamics can be unsafe. Therefore, evaluation must include break scenarios that were not seen during training, or at least that are sampled with low probability during training. The goal is to test whether the controller fails gracefully: reducing exposure, reducing turnover, and staying within constraints.

A critical institutional concept is "graceful failure." No controller can guarantee good performance under arbitrary breaks. But a constraint-respecting controller must guarantee that it does not become reckless. Under break scenarios, performance may drop, but constraints must remain respected. This is the institutional meaning of robustness: preserve capital and compliance even when the model is wrong.

Generalization must also be measured in the presence of execution model error. Execution parameters (spread, impact convexity) can be perturbed. The controller must remain feasible under reasonable cost inflation. If a controller's performance collapses under modest cost inflation, it indicates that the controller's mechanism is not robust and that it may be implicitly exploiting optimistic execution assumptions.

To make generalization auditable, evaluation must report per-seed and per-break outcomes. These are not summarized only by averages. The artifact bundle should include the full set of outcomes, so reviewers can see whether failures cluster under specific breaks. This supports remediation: if the controller fails primarily under correlation compression, concentration constraints or stress training must be strengthened. If it fails under liquidity collapse, turnover and participation constraints must be tightened.

Finally, generalization assessment must be separate from selection. If held-out seeds are used repeatedly during development, they become contaminated: developers implicitly tune to them. Therefore, an institutional pipeline reserves some seeds and break configurations for final acceptance only. This is analogous to keeping an evaluation dataset untouched. In a synthetic lab, it is easy to implement: reserve seed lists and scenario configuration hashes. This discipline is essential to make acceptance credible.

3.6.3 Constraint margins and boundary-time targets

Compliance is not binary. Institutions care about margins. A controller that never breaches but constantly operates at the boundary is not acceptable because operational noise and latency can push it over. Therefore, Chapter 3 defines explicit targets for constraint margins and for boundary time.

Constraint margin is the distance between the realized cost (or risk measure) and the limit. For hard constraints like leverage or max weight, margin can be defined pointwise: $m_t^{\text{lev}} = L_{\max} - L_t$, $m_t^{\text{conc}} = w_{\max} - \max_i w_{t,i}$. For budget constraints like turnover, margin can be defined relative to rolling or episodic aggregates. For tail risk, margin can be defined relative to CVaR budgets. The key is that margin is tracked over time and summarized distributionally.

Boundary time is the fraction of time the system spends within an ϵ-neighborhood of the constraint boundary. For example, boundary time for leverage could be $\frac{1}{T}\sum_t \mathbf{1}\{L_t \geq L_{\max} - \epsilon\}$. Boundary time captures "living at the edge." It is a critical diagnostic because it correlates with fragility. A policy that lives at the boundary is likely to breach under perturbations.

Institutional acceptance therefore includes targets such as: boundary time below a threshold for each hard constraint, and minimum margin quantiles above buffer. For example, require that the 5th percentile margin for leverage remains above a buffer, and boundary time is below 2%. These numbers are illustrative; the key is that they are explicit and predeclared.

Margin targets should be state-dependent. Under stress, margins should increase, not shrink, because uncertainty and execution risk are higher. This is an institutional expectation: de-risk in stress. Therefore, evaluation should compute margins conditional on stress indicators: in high volatility states, in low liquidity states, in crisis regimes. Acceptance should require that margins do not collapse in these states. This prevents controllers that appear safe overall but become dangerous under stress.

Boundary time also connects directly to primal–dual governance signals. If multipliers are high, the system may be pushing the policy away from violations but not away from the boundary. A strong constrained learner should not require extreme multipliers to maintain safe margins. Therefore, acceptance criteria can include multiplier tension metrics and their correlation with boundary time. A controller that maintains margins with moderate multipliers is more naturally aligned.

Finally, margins and boundary time are essential for operationalizing latency-aware buffers. If the internal caps incorporate buffers, then margins measured relative to external caps should be meaningfully positive. This is how we test whether buffers are adequate. If boundary time remains high even with buffers, then either buffers are too small or the policy is too aggressive.

3.6.4 Comparative baselines: conservative + robust MPC as reference

Institutions do not accept a controller because it is sophisticated. They accept it because it is superior to credible alternatives under governance. Therefore, comparative baselines are not optional; they are the anchor of acceptance.

A conservative baseline might be a volatility-targeted exposure controller with drawdown overlays and strict turnover caps. Such a baseline is often simple, transparent, and robust. It may not maximize returns, but it is stable and governable. A constrained learned controller must be compared against this baseline net of costs and under the same scenario packs. If the learned controller cannot beat the baseline meaningfully, there is little justification for additional complexity.

A robust MPC baseline provides a stronger comparator. MPC (Model Predictive Control) uses a short-horizon optimization with explicit constraints. It is naturally aligned with constrained control because it solves a constrained optimization at each step. In a synthetic lab, robust MPC can be implemented as a reference policy that uses conservative forecasts and tight constraints. It is computationally heavier but often more predictable than learned policies. Comparing against robust MPC answers a critical question: do we gain anything from learning beyond what a well-engineered constrained optimizer can do?

Comparisons must be multi-dimensional. It is not enough to compare average returns. We compare: constraint compliance distributions, boundary time, tail metrics, cost dominance, and robustness under breaks. A learned controller that beats baseline returns but increases tail risk or boundary time may be rejected. Conversely, a learned controller that slightly underperforms baseline returns but significantly reduces tail risk may be preferred, depending on mandate. Therefore, the comparative evaluation must be presented as a profile, not a single score.

In institutional practice, this profile comparison is often summarized via a “dominance” view: does the candidate dominate baselines across key dimensions, or does it improve some dimensions while worsening others? If trade-offs exist, they must be explicitly documented and justified. The stage-gate decision is then based on mandate priorities.

Comparative evaluation also supports governance learning. If robust MPC outperforms learned controllers under stress, it suggests that the environment is sufficiently modelable and that explicit constrained optimization may be preferable. If learned controllers outperform robust MPC, it suggests that learning is capturing patterns or adapting in ways MPC cannot, potentially justifying the additional governance cost. Either outcome is informative, and the acceptance pipeline must be capable of producing this evidence.

3.6.5 Advance / do-not-advance decisions as exported artifacts

The final institutional requirement is that acceptance is not an informal conclusion. It is an exported decision artifact that can be reviewed, versioned, and audited. Chapter 3 therefore requires that the notebook produce an explicit stage-gate decision object with clear fields, thresholds, and evidence references.

At minimum, the decision artifact includes: run$_id, confighash, environmentfamilyversion, scenariopackversion, p$
$breachratesforeachconstraint, marginquantiles, boundarytime, tailmetrics, drawdownmetrics, performanceme$

The artifact then provides a recommended action: ADVANCE (to pilot candidate), DO_NOT_ADVANCE (stop), or REVISE_CONTRACT (modify constraints/environment/policy class). It also includes rationale fields: which criteria were binding, which failures were disqualifying, and which remediation suggestions follow from failure signatures. Importantly, the artifact separates facts from interpretation: the metrics are facts; the remediation is interpretation.

Exporting decisions has two governance benefits. First, it prevents narrative selection. If the decision is encoded and validated automatically against thresholds, it is harder to "argue" a strategy into acceptance. Second, it creates a permanent record. Future reviewers can see why a controller was accepted or rejected and can reproduce the evidence.

In a fully governed pipeline, decision artifacts are accompanied by audit bundles: run manifests, training logs including multipliers, evaluation logs, scenario definitions, and policy serialization. The decision artifact points to these files. This creates a traceable chain from decision back to raw evidence. That is the institutional definition of auditability.

3.6.6 Synthesis: acceptance as reproducible evidence, not persuasion

This section defines evaluation and acceptance as reproducible stage gates. Acceptance criteria are distributional: compliance is measured across seeds and scenarios, not in a single run. Generalization is tested across stochastic variability and structural breaks, with emphasis on graceful failure under model error. Constraint margins and boundary time are explicit targets because institutions require buffers, not boundary living. Comparative baselines—conservative controllers and robust MPC—anchor the justification for complexity. Finally, the acceptance outcome is exported as a structured artifact, creating a traceable record that supports committee review and change management.

This completes the institutional logic of Chapter 3: a constrained controller is not accepted because it performed well once, but because it produced auditable evidence of constraint-respecting behavior across uncertainty, with clear superiority or justification relative to robust baselines. The next section will operationalize this acceptance posture in deployment terms: from notebook controller to supervised production candidate, including monitoring, escalation, and change management.

3.7 Operationalization: From Notebook Controller to Supervised Production Candidate

3.7.1 Controller serialization and configuration immutability

If a constrained controller is to be treated as an institutional candidate, it must become a stable, versioned artifact. In research notebooks, it is common to treat “the model” as a set of variables in memory. Institutions cannot do this. They require that the controller be serialized, that its configuration be immutable, and that its provenance be unambiguous. The purpose is not bureaucracy; it is control. A controller that can change without trace is not governable.

Controller serialization means exporting the policy parameters, the policy class metadata, and the governance overlay definition into files that can be loaded and executed deterministically. For a linear controller, this includes the parameter matrix K, bias b, feature normalization parameters, and any structural restrictions. For a nonlinear controller, this includes network weights and architecture metadata. For hybrid controllers, it includes both the learned component and the rule-based overlay parameters. Crucially, serialization must also include the constraint registry: limits, budgets, buffers, and state-dependent tightening rules. Without the constraint registry, the “controller” is incomplete, because the controller is policy plus governance.

Configuration immutability means that the controller is always executed under a configuration that is hashed and signed in the run manifest. All hyperparameters that affect behavior—cost models, scenario pack versions, risk budgets, action projections—must be included in the configuration hash. The system must prevent silent changes. If a change is required, it produces a new version with a new hash and triggers re-evaluation. This is how institutions manage model risk: changes are controlled, reviewed, and approved.

This also implies a clear separation between “parameters learned” and “parameters mandated.” Governance parameters (limits and budgets) are mandated and must not be tuned to improve performance. In an institutional pipeline, these parameters are read from a governance configuration that is separately controlled and approved. The notebook can demonstrate behavior under them, but cannot modify them without generating a new governance version. This separation prevents Goodhart manipulation of supervision settings.

Serialization and immutability also support reproducibility. A reviewer can load the controller artifact, load the configuration, and replay evaluation. This is the operational foundation for acceptance evidence. Without it, the notebook remains a research artifact, not a supervised candidate.

3.7.2 Monitoring: constraint signals, multipliers, and drift flags

A supervised production candidate must be monitorable. Monitoring is not merely logging returns. It is logging the internal variables that indicate whether the controller is drifting toward constraint failure or exploitation. Chapter 3 therefore defines a monitoring design that is consistent with the constrained learning framework.

The primary monitored quantities are constraint signals: drawdown, turnover, participation, tail-risk estimates, concentration, leverage, and any state-dependent tightened caps. These signals must be computed on the same cadence as decisions and must be recorded in a structured log. Monitoring must include both point values and margins: how far are we from limits? This supports early warning. A system that is still compliant but with shrinking margins is approaching a dangerous region.

In a primal–dual framework, multipliers provide additional monitoring. In the notebook, multipliers are updated during training. In a supervised candidate, we may choose to freeze multipliers or update them only under strict controls, but multipliers (or multiplier-like pressure indices) remain valuable telemetry. A rising turnover multiplier indicates that the controller's behavior is "pressing" against turnover budgets. A rising tail multiplier indicates increasing crash vulnerability. These are governance signals that can trigger overlays or human review.

Monitoring must also include drift flags. Drift in this context does not mean statistical drift of features alone. It means drift of behavior and drift of constraint tension. Examples include: increasing boundary time, increasing cost dominance, rising tail loss frequency, changing correlation between risk-on exposures and stress indicators, and divergence between predicted and realized execution costs (slippage mismatch). These flags are particularly important because many failures in trading systems occur when cost models become wrong. Monitoring slippage mismatch is therefore a core institutional control: if realized costs exceed modeled costs persistently, then the controller is operating under false feasibility assumptions.

A key monitoring principle is thresholding and escalation. Monitoring is not useful if it only produces dashboards. It must map to actions: when a flag triggers, what happens? Institutions define escalation pathways: reduce risk, restrict actions, notify supervisors, halt trading. Chapter 3's operationalization therefore treats monitoring thresholds as part of the governance contract. They are versioned and logged, and they define the boundary between autonomy and human supervision.

Finally, monitoring must be auditable. Logs must be structured, timestamped, and tied to controller version. They must be sufficient to reconstruct actions and constraint state. This is essential for post-mortem analysis and for regulatory and internal review.

3.7.3 Escalation pathways: human sign-off and kill-switch logic

No institution deploys an autonomous trading controller without the ability to stop it. Kill-switch logic and escalation pathways are not optional; they are mandatory. The question is not whether escalation exists, but how it is encoded and governed.

In a constrained controller pipeline, escalation triggers can be derived directly from constraint signals and margins. For example: a drawdown trigger above D_{soft} may force a reduction in exposure. A drawdown trigger above D_{hard} may force flattening and require human sign-off to resume. Turnover boundary-time spikes may force throttling of trading. Tail-risk estimate above a threshold may force de-risking or shifting to defensive mode. Slippage mismatch above a threshold may force trading halt because feasibility assumptions are invalid.

Kill-switch logic must be deterministic and simple. Institutions do not want kill switches implemented as learned behaviors. They want hard rules that can be reviewed. Therefore, the governance overlay includes explicit stop-out logic. It should not rely on complex inference. It should be tied to measurable signals.

Human sign-off pathways must be clearly defined. After certain triggers, the system should not automatically resume. It should require review. The review can include verifying whether the trigger was a transient anomaly or a structural change. This is part of institutional accountability: resuming trading after a drawdown event is a decision, not an automatic process.

Escalation pathways should also be tiered. Not all triggers require full shutdown. Some may require tightening caps or switching modes. A hybrid controller architecture is particularly useful here: it can implement discrete modes (normal, cautious, crisis) with different caps. Escalation can shift modes deterministically based on signals. This produces predictable behavior under stress.

All escalation events must be recorded as artifacts: the trigger value, the time, the controller state, and the action taken. This record is essential for audit and for improving governance. Escalation is not an embarrassment; it is a designed behavior.

3.7.4 Change management: what requires re-training and re-approval

Institutions manage models through change control. A supervised candidate controller is not “the model”; it is a versioned artifact with approval scope. Any change outside that scope requires re-evaluation, and often re-approval. Chapter 3 therefore defines change management rules as part of operationalization.

Changes can be categorized as: (i) governance changes (limits, budgets, escalation thresholds), (ii) environment/model changes (execution model parameters, scenario pack definitions, feature computation), (iii) policy changes (parameters or architecture), and (iv) infrastructure changes (data pipelines, logging, seed control). Each category has different approval requirements, but all require traceability.

Governance changes are the most sensitive. Changing a drawdown threshold or turnover cap changes what the institution is willing to tolerate. Such changes must be approved by risk governance and may require committee review. In the surrogate lab, governance changes must produce a new governance version and trigger re-evaluation. It is not acceptable to “tune” governance settings to make performance look better.

Environment and model changes are also critical because they affect feasibility assumptions. If the impact model changes, the controller may no longer be feasible. If scenario packs change, acceptance evidence may no longer apply. Therefore, environment changes require re-running the acceptance suite, at minimum, and may require re-training if learning depended on those assumptions.

Policy changes obviously require re-approval because they change behavior. Even small parameter changes can materially change constraint tension. Therefore, policy changes require a new version and re-evaluation under the full stage-gate criteria.

Infrastructure changes can also invalidate reproducibility. If feature computation changes, state representation changes. If logging changes, auditability changes. Therefore, infrastructure changes require regression tests and may require re-evaluation. Institutions often maintain a “minimum deliverable standard” for artifacts; if infrastructure changes break it, the controller cannot be considered production-ready.

Change management rules should be encoded as checks. For example, if configuration hash changes, the pipeline should automatically label the result as a new version and require acceptance evaluation. This prevents silent drift and supports disciplined deployment.

3.7.5 Documentation pack: what a committee needs to approve a pilot

To approve a pilot deployment, a committee needs a documentation pack that is structured, concise, and evidence-centered. The pack is not an academic report; it is a supervision document. It should contain enough information to judge whether the controller is safe to test with limited capital under oversight.

At minimum, the pack includes: (i) controller description and scope (what it trades, cadence, action space), (ii) governance contract (constraints, budgets, buffers, escalation logic), (iii) training and evaluation evidence (distributional compliance, generalization, stress results), (iv) baseline comparisons, (v) monitoring plan (signals, thresholds, dashboards, alerts), (vi) change management plan, and (vii) limitations and disclaimers (synthetic lab, not validated on real markets, human review required).

The pack should also include the audit bundle references: run manifests, configuration hashes, scenario pack versions, and controller serialization pointers. Committees often do not read raw logs, but they require that logs exist and that they can be inspected if needed. The pack should therefore summarize the evidence and provide traceability.

A critical governance element is explicit acceptance recommendation. The pack should include the stage-gate decision artifact from the notebook. It should state whether the controller is recommended for pilot, and under what conditions: capital limits, exposure limits, monitoring requirements, and stop-out rules. This makes approval conditional and controlled, which is institutionally appropriate.

Finally, the pack should articulate what the pilot is meant to learn. Even in institutional settings, pilots are experiments. They are designed to test feasibility in a more realistic environment. Therefore, the pack should define pilot success metrics and failure triggers. This ensures that deployment is supervised and that outcomes lead to decisions rather than to indefinite operation.

3.7.6 Synthesis: operationalization as the bridge from governed lab to supervised reality

This section translates the constrained controller from a notebook artifact into a supervised production candidate. Serialization and configuration immutability ensure that the controller is a stable, versioned object. Monitoring focuses on constraint signals, margins, and multiplier-like governance telemetry, not just returns. Escalation pathways encode deterministic stop-outs and human sign-off logic. Change management defines what requires re-training and re-approval, preventing silent drift. A committee-facing documentation pack provides evidence, traceability, and conditional approval structure.

Operationalization is the institutional bridge: without it, constrained learning remains a research demonstration. With it, the controller becomes a supervised candidate that can be piloted responsibly. The next section completes Chapter 3 by specifying notebook deliverables: the concrete artifacts the Chapter 3 notebook must generate to implement this operational discipline.

3.8 Conclusion: Why Constrained Surrogacy Is the Institutional Endpoint

3.8.1 What we gain: defensible controllers over impressive backtests

The institutional endpoint of surrogacy is not a clever proxy objective and not an impressive backtest curve. It is a defensible controller: a policy object that can be supervised, stress-tested, audited, and ultimately trusted with bounded autonomy. Chapter 3 completes the escalation from “optimize and test” to “control under mandates.” The gain is a fundamental change in what the organization can claim. Instead of claiming that a strategy performed well in a simulated environment, the institution can claim that a controller behaves within constraints across a distribution of environments and that its acceptance is supported by reproducible evidence.

This matters because institutions do not allocate capital to stories. They allocate capital to systems that can survive review and that can be controlled when the world becomes adversarial. A spectacular backtest often fails this test because it answers the wrong question. It answers "what happened on this path?" rather than "what will we do across many plausible paths when costs rise, correlations compress, signals degrade, and stress becomes the default state?" Constrained surrogacy reframes the objective: performance is pursued only inside a feasible region defined by governance, and feasibility is enforced during learning.

The shift also changes the nature of failure. In backtest-driven development, failure often arrives as a surprise: a strategy "worked" until it did not. In constrained control, failure is surfaced earlier and more cleanly. If constraints are infeasible, multipliers saturate and the stage gate stops. If a policy tries to live on a boundary, boundary-time metrics reveal it. If tail exposure is driving performance, CVaR constraints and stress packs reveal it. This is a professional advantage: the institution learns what it cannot safely do before capital is deployed.

Finally, the defensibility is not only technical; it is organizational. A defended controller is a shared artifact: risk, research, and operations can interpret it in common language—constraints, budgets, margins, triggers, and telemetry. That shared language is what enables institutional adoption. Without it, "AI trading agents" remain boutique prototypes owned by a small group and distrusted by everyone else.

3.8.2 How the proxy contract becomes supervision infrastructure

A proxy objective begins as a compression of economic intent. Chapter 3 shows that, under institutional governance, this compression must mature into a proxy *contract*. The contract is not a single scalar reward. It is a specification with two layers: the performance objective and the constraint system. Once formalized, this contract becomes supervision infrastructure. It defines what the system is allowed to optimize, what it must never violate, and what it must log to make its behavior reviewable.

The CMDP framing is the core of this infrastructure. Reward and constraint costs are separated so that governance settings are not negotiable penalty weights. Hard constraints are enforced by action projections and stop-out transitions. Soft budgets are enforced by primal–dual multipliers that become interpretable governance signals. Robustness is enforced by domain randomization and stress-aware sampling so that the contract holds across plausible worlds. Acceptance is encoded as distributional compliance. When these pieces are implemented, the proxy contract ceases to be a research objective and becomes the institution's operational rulebook embedded in software.

This is an important conceptual outcome: governance is not an afterthought wrapped around an AI agent. It becomes part of the agent's definition. The controller is not "policy only." It is policy plus constraint registry plus enforcement mechanics plus telemetry. The contract also creates organizational accountability. If a controller violates constraints, it is not ambiguous whether this was “acceptable.” The contract defines what is acceptable. If a controller is accepted, the acceptance artifact shows which criteria were met. This is the institutional replacement for persuasion.

The proxy contract also becomes an engineering interface. It standardizes how strategies are developed. Different teams can propose different mechanisms, but they must all satisfy the same constraint registry, the same logging standard, and the same stage-gate criteria. This enables scale. Without a contract, each strategy becomes a bespoke project, and governance becomes inconsistent. With a contract, strategies can be evaluated consistently and compared fairly.

3.8.3 Limits: what remains unverified and why

Even at this institutional endpoint, the work remains a surrogate program. The controller is trained and evaluated in a synthetic environment. It is therefore essential to state what remains unverified. Institutional-grade governance begins with explicit boundaries of knowledge, not with overconfident claims.

First, the mapping from synthetic regimes to real-market regimes is unverified. The synthetic environment may capture structural features—regime switching, volatility clustering, correlation compression, liquidity deterioration—but it cannot guarantee that real markets will match these parameterizations. The purpose of the synthetic lab is not to predict markets; it is to train and test control behaviors under structured uncertainty. The controller’s compliance in synthetic stress does not imply compliance in real stress unless monitoring and escalation systems are correctly implemented and unless real-world feasibility assumptions are validated.

Second, execution realism remains a major uncertainty. Even if convex impact and spread models are included, real execution has microstructure complexities: queue priority, market fragmentation, hidden liquidity, and operational constraints. Synthetic execution is a proxy. Therefore, any claim about net performance is not verified. The institutional value here is not the P&L estimate; it is the disciplined handling of cost sensitivity. The controller is trained to respect turnover and participation limits and to be conservative under liquidity stress. That is transferable. The exact cost numbers are not.

Third, partial observability in real markets is harder than in synthetic labs. Signals degrade, data quality varies, and regimes may not be captured by the modeled belief-state features. Therefore, the state representation is not verified. The institutional approach is to treat the state representation as a bounded operational representation and to require monitoring of drift: if the relationship between features and outcomes changes, governance overlays must tighten or autonomy must be reduced.

Fourth, the acceptance criteria themselves are governance choices. The choice of budgets, percentiles, and thresholds reflects risk appetite. This is not a weakness; it is the nature of institutional control. But it means that “accepted” does not mean “universally safe.” It means “safe under this institution’s mandate and under this tested uncertainty set.” That uncertainty set must be stated and versioned.

These limits are why Chapter 3 ends with a production candidate posture, not with a deployment claim. The endpoint is institutional readiness for supervised pilots, not autonomous release. The system is designed so that it can be monitored, halted, and revised under governance when real-world conditions diverge from assumptions.

3.8.4 How this framework scales to multi-asset, multi-desk mandates

A frequent objection is that constrained surrogacy seems tractable in a toy environment but impossible at institutional scale. Chapter 3's answer is that the framework scales precisely because it is contract-based. Scaling is achieved by standardization of governance interfaces, not by hoping that one model will handle everything.

In a multi-asset setting, the controller's state representation expands to include cross-asset risk summaries, correlation structure estimates, liquidity proxies per asset, and desk-level exposure summaries. The action space becomes higher-dimensional, but the same governance principles apply: bounded actions, action projections onto feasible sets, and constraint registries that include leverage, concentration, turnover, and tail budgets at both asset and portfolio levels. The feasibility surface becomes more complex, but it remains a constrained set.

In a multi-desk setting, constraints become hierarchical. There are desk-level budgets and firm-level budgets. Some constraints are allocated across desks dynamically. The primal–dual framework provides a natural language for this hierarchy: multipliers can be interpreted as internal prices of risk budgets. A firm-level tail-risk multiplier can tighten desk-level budgets during stress. A desk-level liquidity multiplier can tighten trading intensity when local liquidity deteriorates. In other words, dual variables generalize naturally to multi-level governance.

The scenario pack methodology also scales. Scenario packs can be defined at the portfolio level: correlation shocks, liquidity shocks, and regime shifts that affect multiple assets simultaneously. Domain randomization can be defined across market-wide parameters and asset-specific parameters. This allows robust training across plausible cross-asset conditions.

Most importantly, the stage-gate acceptance process scales. Each controller version—whether desk-specific or cross-desk—produces the same artifact bundle: manifests, logs, constraint margins, boundary time, stress outcomes, baseline comparisons, and stage-gate decisions. This standardization enables governance at scale because committees can review controllers in a consistent format rather than learning each system from scratch.

Thus, scaling is not achieved by making one giant black box. It is achieved by making many governed controllers that share the same contract, the same evaluation discipline, and the same monitoring infrastructure.

3.8.5 Roadmap beyond Chapter 3: monitoring, drift, and continuous governance

Chapter 3 is an endpoint in one sense: it completes the methodological escalation from proxy optimization to constrained control. But it is also a beginning, because institutional deployment is not a one-time decision. It is continuous governance under drift.

The roadmap beyond Chapter 3 has three pillars. First is monitoring maturity. Monitoring must evolve from basic constraint logging to predictive early-warning systems: detecting rising constraint tension, rising slippage mismatch, increasing boundary time, and changes in regime prevalence. Monitoring also must be integrated into operational workflows: who receives alerts, how quickly actions are taken, and what actions are permitted automatically versus requiring sign-off.

Second is drift governance. Drift is inevitable: market structure changes, liquidity changes, and strategy impact changes. A governed controller must therefore have drift triggers that reduce autonomy. For example, if realized execution costs exceed modeled costs beyond a tolerance, tighten trading caps. If tail outcomes exceed modeled stress outcomes, tighten risk budgets. If correlation structure deviates, reduce concentration. These responses must be predeclared and versioned. Drift governance is the operational counterpart of robust training: robust training anticipates uncertainty; drift governance reacts when reality exceeds the anticipated set.

Third is continuous evaluation and change control. In production, the acceptance suite becomes a regression suite. New controller versions must be evaluated against baselines and against prior versions. Scenario packs must be updated under governance. Model and infrastructure changes must trigger re-approval. The institution must treat controller evolution like a controlled engineering process, not like ad hoc research.

In this roadmap, constrained surrogacy remains the organizing principle. The system continues to operate within a proxy contract, but that contract is continuously validated and tightened when necessary. The ultimate institutional promise is not perfect prediction. It is controlled behavior under uncertainty, with the ability to supervise, halt, and revise the system when conditions change.

3.8.6 Synthesis: the institutional endpoint is controlled autonomy

Constrained surrogacy is the institutional endpoint because it transforms an AI agent from an optimizer into a supervised controller. We gain defensible behavior profiles rather than seductive backtests. The proxy objective matures into a proxy contract that becomes supervision infrastructure. We remain honest about what is unverified: mapping to real regimes, execution realism, and state representation validity. We also see how the framework scales: by standardizing contracts, constraints, and artifact bundles across assets and desks. Finally, we define the roadmap beyond Chapter 3: monitoring maturity, drift governance, and continuous change control.

The endpoint is therefore best described as *controlled autonomy.* The controller is autonomous within a feasible region, but it is never unsupervised. It is always bounded by mandates, always monitored by telemetry, and always subject to escalation. That is what makes it institutional.

Bibliography

[1] Eitan Altman. *Constrained Markov Decision Processes.* Chapman & Hall/CRC, 1999.

[2] Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming.* Wiley, 1994.

[3] Joshua Achiam, David Held, Aviv Tamar, and Pieter Abbeel. Constrained Policy Optimization. In *Proceedings of the 34th International Conference on Machine Learning (ICML)*, 2017.

[4] Chen Tessler, Daniel J. Mankowitz, and Shie Mannor. Reward Constrained Policy Optimization. In *International Conference on Learning Representations (ICLR)*, 2019.

[5] Yinlam Chow, Mohammad Ghavamzadeh, Lucas Janson, and Marco Pavone. Risk-Constrained Reinforcement Learning with Percentile Risk Criteria. *Journal of Machine Learning Research*, 18(167):1–51, 2017.

[6] R. Tyrrell Rockafellar and Stanislav Uryasev. Optimization of conditional value-at-risk. *Journal of Risk*, 2(3):21–41, 2000.

[7] Dimitris Bertsimas and Melvyn Sim. The price of robustness. *Operations Research*, 52(1):35–53, 2004.

[8] Alberto Bemporad and Manfred Morari. Control of systems integrating logic, dynamics, and constraints. *Automatica*, 35(3):407–427, 1999.

[9] James B. Rawlings, David Q. Mayne, and Moritz M. Diehl. *Model Predictive Control: Theory, Computation, and Design.* Nob Hill Publishing, 2nd edition, 2017.

[10] Robert Almgren and Neil Chriss. Optimal execution of portfolio transactions. *Journal of Risk*, 3(2):5–39, 2001.

Appendix A

Companion Colab Notebook Index

Each chapter is paired with a governed Google Colab notebook implementing a synthetic-first laboratory and producing an auditable run bundle.

Chapter	**Notebook (suggested filename)**
Chapter 1	CH1_PROXY_CONTRACT_LAB.ipynb
Chapter 2	CH2_INSTITUTIONAL_FEASIBILITY_LAB.ipynb
Chapter 3	CH3_CMDP_PRIMAL_DUAL_LAB.ipynb

Appendix B

Minimum Governance Standard for All Labs

Artifact (Save This)

Every companion notebook in this volume must produce:

- Deterministic synthetic generation (seeded) and versioned scenario packs.
- Explicit parameter registry (single source of truth).
- Run manifest (`run_id`, UTC timestamp, environment fingerprint, config hash).
- Prompt / config log (redacted where needed; hashes always).
- Risk log (capability $\uparrow \Rightarrow$ risk $\uparrow \Rightarrow$ controls $\uparrow$).
- Constraint registry (budgets, buffers, tightening rules) and enforcement record.
- Hash ledger for all saved artifacts (plots, logs, serialized controller).
- A zipped audit bundle directory containing `manifest.json`, logs, plots, outputs.
- Clear statement: **Not validated for live deployment; human review required.**

Closing Statement

Institutions do not deploy proxies. They deploy contracts. A surrogate objective without constraints invites exploitation. A backtest without a stage gate invites self-deception.

This mini-book argues for a disciplined endpoint: **constrained surrogacy** — controllers that optimize within mandates, trained under scenario uncertainty, accepted through reproducible evidence, and operated with monitoring, escalation, and change control.

Governance first. Surrogacy second. Feasibility always.