SPEC2TWIN-CHAIN: ORCHESTRATING BI-LEVEL OPTIMIZATION WITH LLMS FOR BLOCKCHAIN DIGITAL TWIN CONSTRUCTION

Anonymous authors
Paper under double-blind review

ABSTRACT

Building a blockchain digital twin requires coupled decisions about simulator structure and parameter calibration. Existing workflows typically begin with a fixed simulator architecture and tune parameters within it, leaving architectural design largely application-specific. Spec2Twin-Chain formulates construction as a bi-level simulation-optimization problem: an LLM worker proposes candidate architectures under a declarative structural specification, a simulation optimizer tunes the variables admitted by each architecture, and evaluated configurations and diagnostics guide later proposal rounds. Controlled experiments using reference evidence from an open-source discrete-event simulator of proof-of-work Ethereum show that the framework constructs and calibrates a structurally valid twin and uses evaluation feedback to improve initial designs with substantial calibration error; supplementary studies reuse the calibrated twin for stress analysis and downstream policy optimization and updating. This division assigns symbolic proposal to the LLM while executable checks, guardrails, and simulation-based optimization determine structural validity and behavioral fit.

1 INTRODUCTION

The behavior of blockchain systems depends on interactions among network conditions, incentive mechanisms, execution environments, and application-layer logic. For instance, transaction-ordering dependencies in decentralized finance (DeFi) have given rise to adversarial strategies that translate application-layer dynamics into consensus-layer risks (Daian et al., 2020; Ethereum Foundation, 2024). Cross-chain bridges also introduce security risks that require careful evaluation before deployment (Belenkov et al., 2025). These examples motivate tools for evaluating emergent behavior under realistic conditions instead of relying only on intuition or limited ad hoc testing. However, at the network layer, the scale and geographic distribution of nodes make faithful live evaluation prohibitively difficult (Aoki et al., 2019). Moreover, because finalized blockchain transactions are *immutable*, direct experimentation on a live chain can incur prohibitive costs and expose participants to irreversible financial losses.

Blockchain digital twins support monitoring and analysis, and let designers explore counterfactual scenarios in a controllable setting. In engineering, a digital twin links a physical system to a virtual representation. However, applying the digital twin concept to blockchains is more challenging, because blockchains emerge from tightly coupled network, consensus, and incentive mechanisms. Transactions and blocks propagate asynchronously, so nodes can temporarily hold different local views of pending transactions and the ledger (Decker and Wattenhofer, 2013), while consensus security and performance depend jointly on network and protocol parameters (Gervais et al., 2016). A blockchain digital twin must therefore represent both system structure and architecture-conditioned operational parameters, consistent with blockchain simulation frameworks such as BlockSim (Alharby and van Moorsel, 2020). Adoption is hindered by the lack of universal reference frameworks, and by the effort required to maintain a twin as the protocol evolves (Sharma et al., 2022). Recent work has used digital twins to manage the trade-off between decentralization, scalability, and security (Diamantopoulos et al., 2022), and has examined Dynamic Data-Driven Application Systems (DDDAS) for blockchain management (Diamantopoulos et al., 2024). Building a blockchain digital twin nevertheless requires substantial manual modeling and domain expertise.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

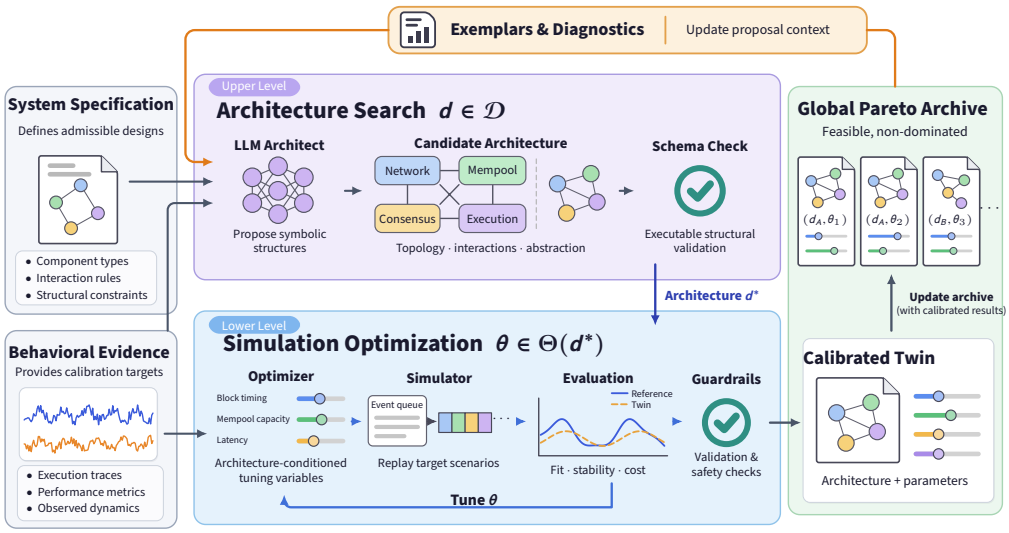


Figure 1: Bi-level optimization framework of *Spec2Twin-Chain*. At the upper level, an LLM architect uses the system specification, behavioral evidence, and feedback context to propose a structurally valid candidate architecture d, including component choices and interaction logic. At the lower level, a simulation-based optimizer tunes the architecture-conditioned variables θ, replays the target scenarios or evidence, evaluates KPI fit, stability, and cost objectives, and enforces guardrail checks. Feasible non-dominated configurations are accumulated in a global Pareto archive; validated exemplars and diagnostics are then fed back to the upper level to guide the next round of architectural search.

Two complementary limitations exist in the conventional digital-twin constructions: bespoke processes (Albshri et al., 2022) require substantial effort on system-specific modeling, whereas LLM-assisted synthesis (Yao et al., 2023) does not guarantee that a plausible proposal is admissible or survives executable validation. To address the modeling challenges (Shao and Helu, 2020; Zargham and Rice, 2019), we propose *Spec2Twin-Chain*, a framework that facilitates blockchain digital-twin development using LLM agents. The workflow proceeds in three steps:

1. LLM agents translate the informal request and its behavioral evidence into the terms of a declarative schema, which specifies the admissible component architectures, parameter constraints, and safety invariants; every candidate twin is verified against this schema. This step allows us to transform blockchain digital-twin development into an optimization problem.

2. The builder then searches the admissible design space through bi-level optimization. At the upper level, an LLM agent proposes/sample candidate architectures concurred with the schema. At the lower level, for each valid proposal, a simulation-based optimizer tunes the variables that the architecture exposes.

3. The upper and lower levels iterate. The scores and diagnostics from each round of the lower level are recorded, and the feasible non-dominated designs form an archive. This archive supplements the context for the subsequent rounds of proposals at the upper level, so that the LLM agents learn from previous experience. Figure 1 summarizes this iterative loop.

Related work is presented in Appendix A.

2 Methodology

Spec2Twin-Chain operates as a *builder* for blockchain digital twins rather than a single, fixed twin instance. Given a target-system specification and behavioral evidence, the method constructs a validated digital-twin configuration by solving a constrained optimization problem. In Section 2.1, we

formulate digital-twin construction as a hierarchical optimization problem over symbolic architectures and architecture-conditioned tuning variables. In Section 2.2, we describe a bi-level solution strategy, where an LLM proposes and refines discrete architectural decisions, while a simulation-based optimizer calibrates the associated tuning vector.

2.1 Optimization Formulation for Digital Twin Construction

Given a target-system specification and behavioral evidence, such as KPI summaries or time-series traces, Spec2Twin-Chain generates a digital-twin configuration that is both structurally valid and behaviorally aligned with the target system. To bridge informal requirements and executable simulation models, we rely on a **declarative structural specification**. This specification describes the admissible design space through component types, interaction patterns, abstraction levels, parameter domains, and safety constraints. Therefore, it guides the search toward structurally coherent digital-twin candidates and filters malformed configurations before expensive simulation.

Methodologically, we formulate twin construction as a constrained optimization problem over this structured space. The decision variables are partitioned into the twin's *architecture* and its architecture-conditioned *tuning variables*. The architecture captures discrete design choices, while the tuning vector captures operational behavior conditional on those choices. Specifically, let $\mathcal{S}$ be the space of candidate twin configurations blue satisfying the schema. A candidate twin $c \in \mathcal{S}$ is represented by

$$c = (d, \theta), \tag{2.1}$$

where:

1. $d \in \mathcal{D}$ denotes the **discrete architectural variables**, in which $\mathcal{D}$ is the admissible architecture set. These variables define the structural composition of the twin, including:
   - *Component topology* $(\mathcal{G})$: the directed graph of interacting modules;
   - *Interaction templates*: the logic governing information, event, or state exchange among the modules;
   - *Fidelity or abstraction modes*: the level of behavioral details that are used to instantiate a component or interaction.
2. $\theta \in \Theta(d)$ denotes the **architecture-conditioned tuning variables**. These variables govern the operational behavior of the instantiated twin, including timing, rates, capacities, thresholds, budgets, protocol limits, and other model-specific controls. Many entries are continuous, but some may be bounded integer or categorical choices when such choices are part of the admissible tuning surface. The notation $\Theta(d)$ emphasizes that the dimensionality, type, and feasible domain of the tuning variables depend on the selected architecture d.

   The practical implementation does not permit an unrestricted LLM-created tuning space. The admissible architecture set $\mathcal{D}$ is quantified by user-supplied requests and declarative system specifications, while final admissibility is further enforced by executable schemas and compiler-owned parameters. For reference, our current implementation contains 83 numeric parameter entries, including 46 canonical entries and 3 discrete entries.

With the decision variables defined, digital-twin construction is formulated as the minimization of a request-conditioned objective vector. Let R be the construction request, including the target specification, behavioral evidence, and success criteria. Denote by $\Omega_R = \{J_1^R, \ldots, J_m^R\}$ the objective set induced by R after these requirements are translated into an executable scoring contract. The optimization problem is

$$\begin{aligned} \min_{\substack{d\in\mathcal{D}\\ \theta\in\Theta(d)}} \quad & \mathbf{J}_R(d,\theta) = \left[J_1^R(d,\theta), \ldots, J_m^R(d,\theta)\right]^\top \\ \text{s.t.} \quad & \mathbb{I}_{\text{ct}}(d,\theta) = 1, \\ & g_k(d,\theta) \le 0, \qquad k \in \mathcal{K}. \end{aligned} \tag{2.2}$$

For vector-valued objective $\mathbf{J}_R$, its minimization problem in (2.2) should be considered as a multi-objective optimization in the Pareto optimal sense. $\mathbb{I}_{\text{ct}}$ is a structural-validity indicator and g_k are dynamic guardrails evaluated during simulation. The formulation is intentionally request-conditioned: different construction tasks may instantiate different objective sets, evidence sources, and guardrail

collections, while retaining the same mathematical structure. We write the problem in minimization form for compactness; objectives that are naturally maximized can be represented through equivalent loss transformations or request-specific objective conventions.

Here, $\mathbf{J}_R$ is generic. Many construction requests draw from three recurring objective families:

1. **Fit discrepancy** (J_{fit}) measures behavioral divergence between digital-twin outputs and reference evidence. A common form is

$$J_{\text{fit}}(c; E) = \|\mathcal{M}(c) - \mathcal{Y}(E)\|, \tag{2.3}$$

   where $\mathcal{M}(c)$ is the vector of simulated performance measures generated by configuration c, and $\mathcal{Y}(E)$ is the evidence-derived target vector under the same measurements. Thus, each *simulated* KPI is compared with the corresponding *evidence* KPI, while $\mathcal{M}$ and $\mathcal{Y}$ distinguish simulator-side measurements from evidence extractions. The norm or distance varies depending on the request.

2. **Stability** (J_{stab}) penalizes configurations that exhibit unrealistic variance, pathological operating regimes, or high failure probability. Stability objectives ensure that a candidate twin does not merely match average behavior, but also remains viable under the relevant operating conditions.

3. **Resource cost** (J_{cost}) captures computational, wall-clock, or modeling burden. This objective family encourages configurations that achieve the desired behavioral fidelity without unnecessary simulation complexity.

These families provide a useful vocabulary for constructing Ω_R, but they are not mandatory. Some construction requests may use all three; others may use a single metric together with stability and feasibility guardrails. The feasible space in (2.2) is restricted by two mechanisms:

1. **Structural validity** $(\mathbb{I}_{\text{ct}})$. This static indicator enforces compliance with the declarative structural specification. This check is performed before simulation and rejects malformed designs, such as incompatible modules or parameters outside admissible bounds.

2. **Dynamic guardrails** (g_k). These constraints represent runtime feasibility requirements monitored during simulation. A violation occurs when $g_k(d, \theta) > 0$. Such candidates are either rejected, or assigned dominated objective values.

2.2 Optimization Strategy: Bi-level Decomposition

Structural field	**Restriction on $\mathcal{D}$ for BlockSim**	**One illustrative architecture d**
Chain profile	Preserve the request's Ethereum proof-of-work and EVM setting.	`chain_profile = evm_pow`.
Module composition	Use registered component types and retain the EVM-PoW mechanisms required by the request.	`network`, `mempool`, `consensus`, `execution`, `fee_market`, `state`, `mining`, and `telemetry`.
Interaction template	Use the default replay template; it is the sole registered template in this repository snapshot.	`default_timeline`.

Table 1: An Illustration of $d \in \mathcal{D}$ for BlockSim simulation, as detailed in Section 3.1 and grounded in the tracked EVM-PoW seed and replay configuration.

The problem in (2.2) is a simulation-based mixed-variable optimization problem over symbolic structures and architecture-conditioned tuning domains. Its structure differs from classical smooth nonlinear programs in three ways:

- **High-dimensional symbolic space.** The architectural variable d is a symbolic configuration defined by component graphs, interaction templates, and abstraction choices. The resulting design space $\mathcal{D}$ is combinatorial and non-Euclidean, making standard relaxations or gradient-based updates unsuitable.

- **Stochastic and expensive black-box evaluation.** The objective vector $\mathbf{J}_R$ is evaluated through simulation. The response surface is generally non-differentiable, non-convex, noisy, and costly to sample.

- **Hierarchical conditionality.** The feasible tuning domain $\Theta(d)$ is conditioned on the architecture. Changing d can change the dimension, interpretation, variable type, and bounds of θ, so the joint space cannot be searched as a single fixed Euclidean box.

These properties motivate a **bi-level solver strategy**, which separates structural search over d from parameter refinement over θ. The original problem (2.2) can be reformulated as:

$$\begin{aligned} \text{(Upper Level)} \quad & \min_{d \in \mathcal{D}} \quad \mathcal{P}(\mathbf{J}_R(d, \theta^*(d))) \\ \text{s.t.} \quad & \mathbb{I}_{\text{ct}}(d) = 1, \\ \text{(Lower Level)} \quad & \theta^*(d) \in \arg\min_{\theta \in \Theta(d)} \mathbf{J}_R(d, \theta) \\ \text{s.t.} \quad & g_k(d, \theta) \leq 0, \qquad k \in \mathcal{K}. \end{aligned} \tag{2.4}$$

Here, $\theta^*(d)$ denotes the optimized tuning outcome for a fixed architecture d; in the multi-objective setting, this may correspond to a non-dominated set rather than a single vector. The mapping $\mathcal{P}(\cdot)$ denotes the Pareto frontier, or a request-specific summary of that frontier. Thus, the upper level compares admissible architectures by the quality of the optimized trade-off surface they induce, while the lower level searches for feasible and non-dominated parameterizations within each architecture-conditioned domain. With a slight abuse of notation, $\mathbb{I}_{\text{ct}}(d) = 1$ denotes the architecture-level checks, which do not depend on θ; the θ-level domain condition is enforced within the lower level.

However, (2.4) does not require exhaustive optimization over every $d \in \mathcal{D}$: the lower-level problem is solved only for the finite set of architectures proposed by the upper level and *accepted by structural validation*. For example, Table 1 illustrates selected structural fields of a BlockSim-compatible architecture d, using the EVM-PoW family seed and default replay configuration. The admissible set $\mathcal{D}$ only consists of architecture specifications satisfying the construction request and executable structural checks. For this fixed d, block timing, transaction demand, mempool capacity, and network latency belong to the lower-level tuning vector $\theta \in \Theta(d)$.

The resulting nested solver architecture bridges semantic reasoning with simulation-based parameter search: bi-level optimization provides a principled separation between design selection and parameter calibration (Colson et al., 2007), while tool-using scientific-agent systems show that LLMs are more reliable when they plan, retrieve, and orchestrate external tools rather than replace domain simulators or optimizers (Yao et al., 2023; Boiko et al., 2023; Bran et al., 2024). See Appendix B for a detailed description on the two levels and the feedback mechanism that links them.

Algorithm 1 Spec2Twin-Chain Bi-Level Optimization Loop

Require: Request R with target specification $\mathfrak{S}$ and behavioral evidence E; upper-level iterations T; candidate budget N

$\mathcal{A}_{\text{global}} \leftarrow \emptyset$, $H_0 \leftarrow \emptyset$
for $t = 1$ **to** T **do**
  Infer or update objective contract $\mathbf{J}_R$ and guardrails from R and H_{t-1}
  Generate candidate architectures $\mathcal{D}_t = \{d_t^{(1)}, \ldots, d_t^{(N)}\}$ using the LLM architect
  Retain candidates satisfying $\mathbb{I}_{\text{ct}}$
  for all valid $d \in \mathcal{D}_t$ **do**
    $\mathcal{P}_d^* \leftarrow \texttt{SimOpt}(d, \mathbf{J}_R, \Theta(d), \{g_k\}_{k \in \mathcal{K}})$
    $\mathcal{A}_{\text{global}} \leftarrow \texttt{NonDominatedMerge}(\mathcal{A}_{\text{global}}, \mathcal{P}_d^*)$
  end for
  Select representative exemplars $\mathcal{E}_t \subset \mathcal{A}_{\text{global}}$
  $H_t \leftarrow \texttt{FormatFeedback}(\mathcal{E}_t)$
end for
return $\mathcal{A}_{\text{global}}$

3 EXPERIMENTS AND EVALUATION

We now present the numerical experiments, beginning with background on the blockchain setting for which the digital twins are built. Ethereum stages pending transactions in node-local transaction pools; these overlapping views, propagated through transaction gossip, are commonly referred to as the public mempool. A user broadcasts a transaction, it spreads across the peer-to-peer network, and it waits among other pending transactions until a block includes it. Under proof of stake, Ethereum schedules a proposer for each 12-second slot (Ethereum Consensus Specifications contributors, 2026). During this wait, block construction and the fee market jointly determine whether and when a transaction is included, how it is ordered, and the effective fee it pays (Buterin et al., 2019; Roughgarden, 2020). Ordering carries value of its own: a searcher that observes a pending trade can profit by submitting a competing transaction that executes ahead of it, a practice documented at scale on decentralized exchanges (Daian et al., 2020).

This structure generates several of the protocol design problems that occupy current Ethereum research. Pending transactions in the public mempool are visible before execution, so they can be front-run, and economically valuable order flow is therefore often routed through private channels instead. The value extractable by including, excluding, or reordering transactions is known as maximal extractable value (MEV) (Ethereum.org contributors, 2026a). MEV has motivated an out-of-protocol form of proposer-builder separation: participating validators obtain externally constructed payloads from specialized builders, which compete by submitting bids through MEV-Boost relays. Post-Merge measurement studies found that this builder-mediated path rapidly became the dominant way blocks reach the chain (Wahrstätter et al., 2023; Yang et al., 2025). Countermeasures such as encrypted mempools and inclusion lists remain under debate (Ethereum.org contributors, 2026b). Evaluating any of these designs on the live network can have direct financial consequences, and the surrounding strategic behavior cannot be held fixed for comparison, which is what makes controlled experimentation on a calibrated model attractive.

Our evaluation targets this setting in two parts. Part 1 examines whether the builder can produce a structurally valid, quantitatively calibrated mempool twin from a system description and predetermined behavioral evidence, including target KPIs. Part 2 studies congestion (Part 2a) and adversarial stress (Part 2b) using the calibrated twin. Supplementary policy-optimization studies, MEV-style block construction (Part 2.1a) and sender-side fee and replacement policies (Part 2.1b), are presented in Appendix C.1; policy adaptation under drift (Part 2.2) is presented in Appendix C.2. These controlled scenarios demonstrate the bi-level methodology rather than establish live-chain validity. This organization separates calibration validity from downstream usefulness.

For Part 1, we use BlockSim as the reference simulator. Introduced by (Alharby and van Moorsel, 2020), this open-source discrete-event simulator provides configurable network, consensus, and incentive models, together with controlled KPI targets and traces for calibration.[1] Its configurable parameters support scenario generation without the data shortages that subsampling observational records may entail. We also test selective evidence feeding under sparse outer-loop data, illustrating a workflow applicable to real-world evidence retrieved by the LLM. Transfer to other settings requires corresponding modules, evidence, and validation. Other potential evidence sources include SimBlock for block-propagation studies (Aoki et al., 2019) and BlockEmulator for blockchain-sharding protocol emulation (Huang et al., 2025), provided their abstractions match the target specification.

Across all experiments, the upper-level design step is implemented with Codex-based LLM workers. An *LLM worker* is a model instance that authors a candidate architecture, policy, or skeleton conditioned on three experiment-specific objects:

(a) a schema, which instantiates the structural-validity condition I_{ct}, determines the admissible architecture set $\mathcal{D}$ and architecture-conditioned tuning domains $\{\Theta(d)\}_{d\in\mathcal{D}}$;

(b) an objective contract, which instantiates $\mathbf{J}_R$, its own objective directions and weights, and the guardrails $\{g_k\}$;

(c) an evidence pack, which instantiates E.

[1] BlockSim GitHub repository: `https://github.com/maher243/BlockSim`.

Each accepted upper-level candidate is counted as a *Codex-authored proposal*. Proposal counts refer to upper-level candidate architectures or policies that are passed to the lower-level optimizer for evaluation. Based on our implementation, for the reported Part 2 runs, `proposal_count` = `algorithm_iterations` × `candidate_budget_per_iteration`. Importantly, the count does not include raw request-processing calls or within-candidate repair turns.

For each reported experiment, the declarative *schema* fixes the admissibility rules for $\mathcal{D}$ and the architecture-conditioned domains $\Theta(d)$. The upper level therefore explores different d within a fixed admissible universe. Separately, the *objective contract* is instantiated before search and held fixed across upper-level iterations, so archive dominance is always evaluated under a single contract.

Table 3 summarizes the six experimental instances. In each case, the upper level specifies a discrete architecture, policy, or update-scope structure d, and the lower level optimizes architecture-conditioned tuning variables θ under objective and guardrail contracts. Appendix D provides the experiment details and additional results: runtime and reproducibility settings are reported in Section D.1; Part 1 KPI and feedback-ablation evidence is reported in Section D.2; Part 2.1 stress-regime diagnostics are reported in Section D.3; Part 2.2 policy-optimization details are reported in Section D.4; and Part 2.3 update-policy details are reported in Section D.5.

3.1 Part 1: BlockSim-Grounded Digital-Twin Calibration

The first experiment evaluates the core construction capability of Spec2Twin-Chain. The target is an Ethereum proof-of-work mempool setting, and the evidence pack contains BlockSim configuration metadata, execution traces, and ten KPI targets including block production, transaction throughput, gas usage, uncle generation[2], and block-time distribution. The reported evidence horizon contains 33 blocks and 4,534 transactions, corresponding to an average of 137.394 transactions per block. The ten KPIs in Figure 2 are aggregate statistics extracted from this finite horizon rather than a single block-count target. The upper-level worker constructs an EVM-PoW architecture with mempool, consensus, mining, network, execution, fee-market, state, and telemetry modules. The lower level tunes block timing, mining, gas, transaction-arrival, mempool-capacity, latency, and execution parameters within evidence-guided bounds.

Figure 2 shows the KPI-level error profile. The generated twin matches the count and throughput metrics closely: block count, transaction count, transactions per block, uncle count, uncle rate, gas usage, and TPS have zero or negligible relative error. The remaining error is concentrated in timing metrics, especially average block time. This pattern is consistent with the modeling role of the twin: the builder recovers aggregate throughput, gas, and uncle behavior while approximating the stochastic timing distribution through a lower-dimensional parameterization.

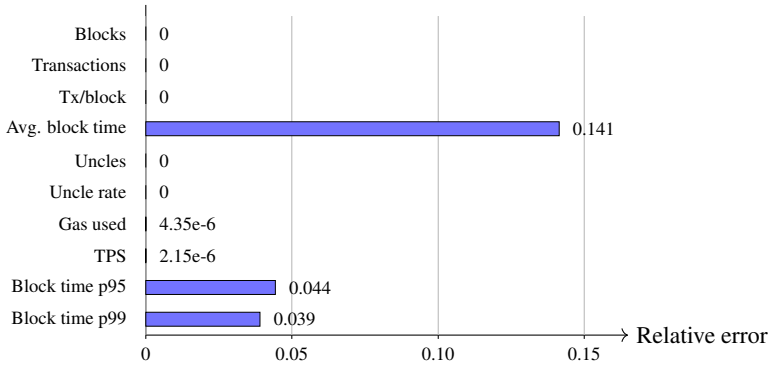


Figure 2: KPI-level calibration error for the full-evidence BlockSim calibration run. The generated digital twin matches block count, transaction count, average transactions per block, uncle count, uncle rate, gas usage, and throughput with negligible error. The remaining aggregate fit error is concentrated in block-time distribution metrics, especially average block time.

[2]An uncle (ommer) in pre-Merge proof-of-work Ethereum is a valid competing block that does not enter the canonical chain but may be referenced by a later canonical block and receive a partial reward.

The aggregate fit error is 0.02248, below the target threshold of 0.05. The hard runtime guardrail is also satisfied because the calibrated average block time is 13.0683 seconds, below the 20-second limit. Therefore, the full-evidence calibration study supports the first claim: the LLM-guided upper level can produce a schema-valid architecture and scoring contract, and the lower-level optimizer can calibrate the associated tuning variables to controlled simulator evidence.

We also test whether feedback helps when the initial request is less informative. Without evidence-guided bounds, the first candidate has a fit error of 0.30079, but the loop recovers to 0.02248 after diagnostic feedback. Under progressive feedback, the error remains high in the first two iterations, improves after partial diagnostics, and falls below the threshold once full diagnostics are available. Detailed KPI table and ablation traces in Appendix D.2 (Table 7, Figure 6, and Table 8) show that the feedback archive is not merely a log of previous evaluations; it supplies actionable signals for revising later upper-level proposals and lower-level search contracts.

Overall, Part 1 shows that Spec2Twin-Chain can construct a calibrated EVM-PoW mempool digital twin against controlled BlockSim evidence. The claim is intentionally scoped: the experiment does not imply post-Merge Ethereum fidelity, PBS or MEV-Boost fidelity, or byte-for-byte equivalence with BlockSim.

3.2 Part 2: Stress-Regime Analysis with the Calibrated Mempool Twin

In Part 1, the tuning variables serve as calibration parameters that reproduce reference behavior, rather than controls that can be freely adjusted on the underlying blockchain. Once calibrated, the twin provides a controllable environment for exploring hypothetical conditions beyond those represented in the reference evidence. Part 2 therefore shifts from reproducing baseline behavior to characterizing stress responses. This enables controlled what-if analysis without intervening on the live system.

We consider two stress-regime scenarios. Part 2a uses the twin as a congestion and scaling what-if engine, while Part 2b uses it as an adversarial-stress substrate. In both studies, the upper level proposes a stress-specific architecture and observation surface, and the lower level tunes architecture-conditioned parameters under scenario-specific objectives and guardrails. Table 2 reports the compact result summary; the corresponding run-level objectives, scenario thresholds, and adversarial diagnostics are reported in Appendix D.3, Tables 9–11 and Figure 7.

Table 2: Part 2 stress-regime results over the calibrated mempool twin.

Study	Proposals	Fallbacks	Primary result	Interpretation
Part 2a: Congestion	32	0	Zero throughput-fit error; conservative breakdown threshold of 76 TPS; no divergence or eviction.	The congestion twin exposes queue and latency stress while satisfying guardrails.
Part 2b: Adversarial stress	6	0	Zero profile-fit error; no guardrail violations; strict fingerprint match rate of 0.4.	The adversarial twin matches aggregate stress profiles but exposes incomplete class-specific fingerprints.

3.2.1 Part 2a: Congestion-Regime What-if Analysis

The congestion study contains three deterministic, schema-validated stress fixtures: a peak-load TPS spike, a gas-spike trace, and a throughput-collapse trace. At each tick, a fixture specifies offered demand (`demand_tps`), block capacity (`block_capacity_tps`), baseline latency (`base_latency_ms`), and a gas multiplier (`gas_multiplier`), together with a reference throughput and guardrails. The evaluator computes the adjusted capacity as $(C_t / \max(0.25, g_t))$, propagates queue, and defines throughput fit as RMS error against the reference curve. A scenario's breakdown threshold is the first raw offered-demand point once queue depth reaches the optimized queue threshold or the backpressure signal turns positive. The Codex-authored congestion twin represents a queue-mediated service-compression mechanism: elevated arrivals build backlog, backlog increases fee competition and latency, and the combined pressure reduces effective inclusion capacity. Mixed-variable NSGA-II then tunes five parameters: traffic-rate multiplier, queue threshold, backpressure coefficient, elasticity exponent, and latency sensitivity.

Figure 3 visualizes the resulting stress trajectories. The gas-spike trace gives the most conservative breakdown threshold, 76 TPS, because throughput drops while queue and latency increase. The throughput-collapse scenario produces the largest queue, 292.4095 transactions, and the peak latency, 258.8642 ms. Thus, the result is not only an aggregate objective match; it also provides an interpretable congestion profile under controlled stress conditions.

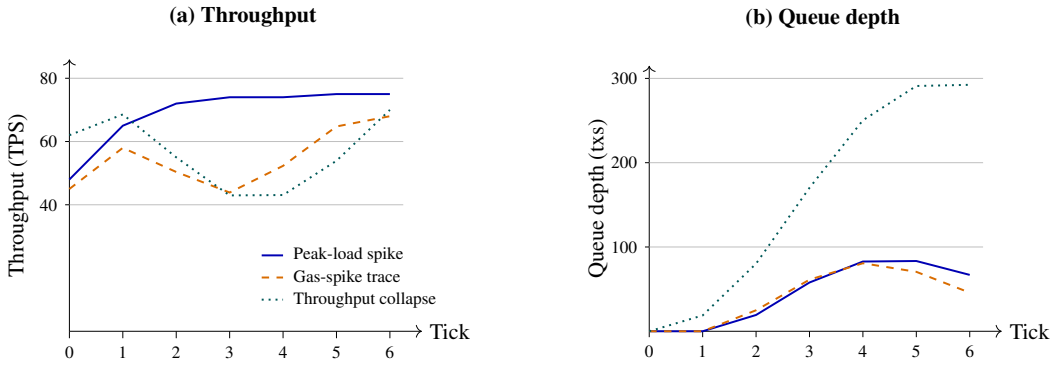


Figure 3: Part 2a congestion trajectories for the congestion what-if twin.

The Part 2a evidence supports a downstream-use claim: after calibration, Spec2Twin-Chain can reuse the mempool twin as a what-if substrate for congestion-regime analysis. The reported TPS threshold is a scenario-specific diagnostic rather than a production-chain capacity guarantee.

3.2.2 Part 2b: Adversarial Stress Analysis

Part 2b constructs a joint mempool–adversary stress twin over five controlled attack classes: mempool spam, frontrunning injection, eclipse-style network impairment, selfish-mining timing, and covert transaction replacement. The upper-level variable defines attack channels, targets, observation surfaces, fingerprint logic, detection gates, and stability guardrails. The lower level tunes adversary budget, timing, concealment, disruption, and detection parameters.

The run reaches zero aggregate profile-fit error and no guardrail violations. However, only two of the five strict fingerprints match the scenario oracle: mempool spam and frontrunning injection. Eclipse, selfish-timing, and covert-replacement cases remain too covert or too slowly rising under the scenario criteria. This is a useful limitation rather than a hidden failure, because it separates profile-level matching from class-signature recovery. Additional high-budget probes did not improve the fingerprint rate or stability term and increased cost in one probe. Therefore, the evidence does not indicate that the remaining mismatch is only a search-budget shortage.

Overall, Part 2 shows that the calibrated twin can be transformed into stress-analysis twins that remain executable and guardrail-satisfying. The claims remain limited to controlled scenario analysis rather than live-chain adversarial validation.

4 Conclusion and Future Work

Spec2Twin-Chain treats digital-twin structure and calibration as coupled design choices and searches over both. LLM workers propose executable simulator structures, while numerical solvers tune the parameters available in each structure. A proposal is retained only after it passes schema checks, guardrails, and simulation tests. In the BlockSim study, this process produces a valid mempool twin that matches the reference behavior, and evaluation feedback helps the worker correct an initial design with substantial calibration error.

The next question is whether this construction process transfers to other blockchain systems and evidence sources. Applying the process to a new target entails a corresponding structural specification and validation contract. Higher-fidelity simulators also increase the cost of architecture search. Surrogate models or adaptive evaluation budgets may help the lower level spend simulation runs where they are most informative while preserving executable validation of every proposed structure.

AI USE STATEMENT

We used OpenAI Codex to assist with code development and language polishing of the manuscript. Codex also helped identify relevant literature. AI-assisted code was tested, and suggested references were checked against the original sources. The authors take responsibility for the final manuscript, code, and reported results.

REPRODUCIBILITY STATEMENT

The paper and appendices describe the bi-level procedure, experimental settings, and evaluation protocols. Our implementation, experiment scripts, result summaries, and verification scripts are available in an anonymous repository at `https://anonymous.4open.science/r/blockchain-dt-review-30E5/`.

REFERENCES

Adams, H., Zinsmeister, N., Salem, M., Keefer, R., and Robinson, D. (2021). Uniswap v3 core. Uniswap Labs White Paper.

Albshri, A., Alzubaidi, A., Awaji, B., and Solaiman, E. (2022). Blockchain simulators: A systematic mapping study. In *2022 IEEE International Conference on Services Computing (SCC)*, pages 284–294. IEEE.

Alharby, M. and van Moorsel, A. (2020). Blocksim: An extensible simulation tool for blockchain systems. *Frontiers in Blockchain*, 3:28.

Aoki, Y., Otsuki, K., Kaneko, T., Banno, R., and Shudo, K. (2019). Simblock: A blockchain network simulator. In *IEEE INFOCOM 2019–IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 325–329. IEEE.

Azaria, A., Ekblaw, A., Vieira, T., and Lippman, A. (2016). Medrec: Using blockchain for medical data access and permission management. In *2016 2nd International Conference on Open and Big Data (OBD)*, pages 25–30.

Belenkov, N., Callens, V., Murashkin, A., Bak, K., Derka, M., Gorzny, J., and Lee, S.-S. (2025). Sok: A review of cross-chain bridge hacks in 2023. *arXiv preprint arXiv:2501.03423*.

Bifrost (2022). Post-mortem: Bifi-btc illegal address registration. `https://bifrost.medium.com/post-mortem-bifi-btc-illegal-address-registration-c21ce3ba9fc8`. Accessed 2026-02-09.

Böhme, R., Christin, N., Edelman, B., and Moore, T. (2015). Bitcoin: Economics, technology, and governance. *Journal of Economic Perspectives*, 29(2):213–238.

Boiko, D. A., MacKnight, R., Kline, B., and Gomes, G. (2023). Autonomous chemical research with large language models. *Nature*, 624(7992):570–578.

Bran, A. M., Cox, S., Schilter, O., Baldassari, C., White, A. D., and Schwaller, P. (2024). Augmenting large language models with chemistry tools. *Nature Machine Intelligence*, 6(5):525–535.

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., and Amodei, D. (2020). Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901.

Buterin, V., Conner, E., Dudley, R., Slipper, M., Norden, I., and Bakhta, A. (2019). EIP-1559: Fee market change for ETH 1.0 chain. Ethereum Improvement Proposals, no. 1559. `https://eips.ethereum.org/EIPS/eip-1559`. Accessed July 14, 2026.

Chen, Z.-Y., Shen, S., Shen, G., Zhi, G., Chen, X., and Lin, Y. (2024). Towards tool use alignment of large language models. In Al-Onaizan, Y., Bansal, M., and Chen, Y.-N., editors, *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 1382–1400, Miami, Florida, USA. Association for Computational Linguistics.

Colson, B., Marcotte, P., and Savard, G. (2007). An overview of bilevel optimization. *Annals of Operations Research*, 153(1):235–256.

CSIRO (2022). Certifying future minerals trade. Commonwealth Scientific and Industrial Research Organisation. Accessed 23 May 2026.

Daian, P., Goldfeder, S., Kell, T., Li, Y., Zhao, X., Bentov, I., Breidenbach, L., and Juels, A. (2020). Flash boys 2.0: Frontrunning in decentralized exchanges, miner extractable value, and consensus instability. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 910–927. IEEE.

Deb, K., Pratap, A., Agarwal, S., and Meyarivan, T. (2002). A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197.

Decker, C. and Wattenhofer, R. (2013). Information propagation in the bitcoin network. In *2013 IEEE Thirteenth International Conference on Peer-to-Peer Computing (P2P)*, pages 1–10. IEEE.

Diamantopoulos, G., Tziritas, N., Bahsoon, R., and Theodoropoulos, G. (2022). Digital twins for dynamic management of blockchain systems. In *2022 Winter Simulation Conference (WSC)*, pages 2876–2887. IEEE.

Diamantopoulos, G., Tziritas, N., Bahsoon, R., and Theodoropoulos, G. (2024). Dynamic data-driven digital twins for blockchain systems. In *Dynamic Data Driven Applications Systems*, Lecture Notes in Computer Science, pages 283–292. Springer.

Dinh, T. T. A., Wang, J., Chen, G., Liu, R., Ooi, B. C., and Tan, K.-L. (2017). BLOCKBENCH: A framework for analyzing private blockchains. In *Proceedings of the 2017 ACM International Conference on Management of Data*, pages 1085–1100. Association for Computing Machinery.

Dong, Q., Li, L., Dai, D., Zheng, C., Ma, J., Li, R., Xia, H., Xu, J., Wu, Z., Liu, T., Chang, B., Sun, X., Li, L., and Sui, Z. (2023). A survey on in-context learning. *arXiv preprint arXiv:2301.00234*.

Ethereum Consensus Specifications contributors (2026). Ethereum proof-of-stake consensus specifications: Beacon chain. Ethereum Consensus Specifications. `https://ethereum.github.io/consensus-specs/phase0/beacon-chain/`. Accessed July 14, 2026.

Ethereum Foundation (2024). Proposer-builder separation. `https://www.ethereum.org/en/roadmap/pbs/`.

Ethereum.org contributors (2026a). Maximal extractable value (MEV). Ethereum.org Developer Documentation. `https://ethereum.org/developers/docs/mev/`. Accessed July 14, 2026.

Ethereum.org contributors (2026b). Proposer-builder separation. Ethereum.org Roadmap. `https://ethereum.org/roadmap/pbs/`. Accessed July 14, 2026.

Forrester, A. I. J., Sóbester, A., and Keane, A. J. (2007). Multi-fidelity optimization via surrogate modelling. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 463(2088):3251–3269.

Gervais, A., Karame, G. O., Wüst, K., Glykantzis, V., Ritzdorf, H., and Capkun, S. (2016). On the security and performance of proof of work blockchains. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, pages 3–16.

Glasserman, P. and Yao, D. D. (1992). Generalized semi-markov processes: Antimatroid structure and second-order properties. *Mathematics of Operations Research*, 17(2):444–469.

Göbel, J., Keeler, H. P., Krzesinski, A. E., and Taylor, P. G. (2016). Bitcoin blockchain dynamics: the selfish-mine strategy in the presence of propagation delay. *Performance Evaluation*, 104:23–41.

Hansen, N. (2016). The CMA evolution strategy: A tutorial. *arXiv preprint arXiv:1604.00772*.

Huang, H., Ye, G., Yang, Q., Chen, Q., Yin, Z., Luo, X., Lin, J., Zheng, J., Li, T., and Zheng, Z. (2025). Blockemulator: An emulator enabling to test blockchain sharding protocols. *IEEE Transactions on Services Computing*, 18(2):690–703.

Huang, X., Liu, W., Chen, X., Wang, X., Wang, H., Lian, D., Wang, Y., Tang, R., and Chen, E. (2024). Understanding the planning of LLM agents: A survey. *arXiv preprint arXiv:2402.02716*.

Hyperliquid (2026a). About hyperliquid. Hyperliquid Documentation. `https://hyperliquid.gitbook.io/hyperliquid-docs`. Accessed August 21, 2026.

Hyperliquid (2026b). Order book. Hyperliquid Documentation. `https://hyperliquid.gitbook.io/hyperliquid-docs/hypercore/order-book`. Accessed August 21, 2026.

IBM (2018). Reimagining the global food supply through blockchain: Q&a with brigid mcdermott. IBM Newsroom. Accessed 23 May 2026.

Kawaguchi, N. (2019). Application of blockchain to supply chain: Flexible blockchain technology. *Procedia Computer Science*, 164:143–148.

Khan, I., Majib, Y., Ullah, R., and Rana, O. (2024). Blockchain applications for internet of things—a survey. *Internet of Things*, 27:101254.

Krishnamoorthy, S., Mashkaria, S. M., and Grover, A. (2023). Diffusion models for black-box optimization. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 17842–17857. PMLR.

Kshetri, N. (2022). Blockchain systems and ethical sourcing in the mineral and metal industry: a multiple case study. *The International Journal of Logistics Management*, 33(1):1–27.

Kumar, A. and Levine, S. (2020). Model inversion networks for model-based optimization. In *Advances in Neural Information Processing Systems*, volume 33, pages 5126–5137.

Laszka, A., Eisele, S., Dubey, A., Karsai, G., and Kvaternik, K. (2018). Transax: A blockchain-based decentralized forward-trading energy exchanged for transactive microgrids. In *2018 IEEE 24th International Conference on Parallel and Distributed Systems (ICPADS)*, pages 918–927. IEEE.

Lee, S.-S., Murashkin, A., Derka, M., and Gorzny, J. (2023). SoK: Not quite water under the bridge: Review of cross-chain bridge hacks. In *2023 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, pages 1–14. IEEE.

Li, G., Hammoud, H. A. A. K., Itani, H., Khizbullin, D., and Ghanem, B. (2023). CAMEL: Communicative agents for "mind" exploration of large language model society. In *Advances in Neural Information Processing Systems*, volume 36.

Li, W., Bu, J., Li, X., and Chen, X. (2022). Security analysis of defi: Vulnerabilities, attacks and advances. In *2022 IEEE International Conference on Blockchain (Blockchain)*, pages 488–493. IEEE.

Li, Z., Yuan, H., Huang, K., Ni, C., Ye, Y., Chen, M., and Wang, M. (2024). Diffusion model for data-driven black-box optimization. *arXiv preprint arXiv:2403.13219*.

Min, S., Lyu, X., Holtzman, A., Artetxe, M., Lewis, M., Hajishirzi, H., and Zettlemoyer, L. (2022). Rethinking the role of demonstrations: What makes in-context learning work? In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 11048–11064, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Nasrulin, B., De Vos, M., Ishmaev, G., and Pouwelse, J. (2022). Gromit: Benchmarking the performance and scalability of blockchain systems. In *2022 IEEE International Conference on Decentralized Applications and Infrastructures (DAPPS)*, pages 56–63.

Neudecker, T., Andelfinger, P., and Hartenstein, H. (2015). A simulation model for analysis of attacks on the bitcoin peer-to-peer network. In *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, pages 1327–1332. IEEE.

Park, J. S., O'Brien, J. C., Cai, C. J., Morris, M. R., Liang, P., and Bernstein, M. S. (2023). Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*, pages 1–22. Association for Computing Machinery.

Patil, S. G., Zhang, T., Wang, X., and Gonzalez, J. E. (2024). Gorilla: Large language model connected with massive apis. In *Advances in Neural Information Processing Systems*, volume 37.

QANplatform (2022). Qanx bridge disclosure analysis (continuously updated). https://medium.com/qanplatform/qanx-bridge-disclosure-analysis-continuously-updated-849f7a91d05e. Accessed 2026-02-09.

Qian, C., Liu, W., Liu, H., Chen, N., Dang, Y., Li, J., Yang, C., Chen, W., Su, Y., Cong, X., Xu, J., Li, D., Liu, Z., and Sun, M. (2024). Chatdev: Communicative agents for software development. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 15174–15186. Association for Computational Linguistics.

Qin, Y., Liang, S., Ye, Y., Zhu, K., Yan, L., Lu, Y., Lin, Y., Cong, X., Tang, X., Qian, B., Zhao, S., Hong, L., Tian, R., Xie, R., Zhou, J., Gerstein, M., Li, D., Liu, Z., and Sun, M. (2023). Toolllm: Facilitating large language models to master 16000+ real-world apis. *arXiv preprint arXiv:2307.16789*.

Rekt (2021). Poly network - rekt. https://rekt.news/polynetwork-rekt/. Accessed 2026-02-09.

Rekt (2022a). Ankr & helio - rekt. https://rekt.news/ankr-helio-rekt/. Accessed 2026-02-09.

Rekt (2022b). Wormhole - rekt. https://rekt.news/wormhole-rekt/. Accessed 2026-02-09.

Roughgarden, T. (2020). Transaction fee mechanism design for the Ethereum blockchain: An economic analysis of EIP-1559. arXiv preprint arXiv:2012.00854.

Schick, T., Dwivedi-Yu, J., Dessì, R., Raileanu, R., Lomeli, M., Hambro, E., Zettlemoyer, L., Cancedda, N., and Scialom, T. (2023). Toolformer: Language models can teach themselves to use tools. In *Advances in Neural Information Processing Systems*, volume 36.

Shahriari, B., Swersky, K., Wang, Z., Adams, R. P., and de Freitas, N. (2016). Taking the human out of the loop: A review of bayesian optimization. *Proceedings of the IEEE*, 104(1):148–175.

Shao, G. and Helu, M. (2020). Framework for a digital twin in manufacturing: Scope and requirements. *Manufacturing Letters*, 24:105–107.

Sharma, A., Kosasih, E., Zhang, J., Brintrup, A., and Calinescu, A. (2022). Digital twins: State of the art theory and practice, challenges, and open research questions. *Journal of Industrial Information Integration*, 30:100383.

Shinn, N., Cassano, F., Gopinath, A., Narasimhan, K., and Yao, S. (2023). Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 36.

Snoek, J., Larochelle, H., and Adams, R. P. (2012). Practical bayesian optimization of machine learning algorithms. In *Advances in Neural Information Processing Systems*, volume 25, pages 2951–2959.

Tan, K.-C. (2002). Supply chain management: Practices, concerns, and performance issues. *Journal of Supply Chain Management*, 38(4):42–53.

Teng, F., Zhang, Q., Wang, G., Liu, J., and Li, H. (2021). A comprehensive review of energy blockchain: Application scenarios and development trends. *International Journal of Energy Research*, 45(12):17515–17531.

Wahrstätter, A., Zhou, L., Qin, K., Svetinovic, D., and Gervais, A. (2023). Time to bribe: Measuring block construction markets. Cryptology ePrint Archive, Paper 2023/760.

Wang, G., Xie, Y., Jiang, Y., Mandlekar, A., Xiao, C., Zhu, Y., Fan, L., and Anandkumar, A. (2023). Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*.

Wang, L., Ma, C., Feng, X., Zhang, Z., Yang, H., Zhang, J., Chen, Z., Tang, J., Chen, X., Lin, Y., Zhao, W. X., Wei, Z., and Wen, J.-R. (2024). A survey on large language model based autonomous agents. *Frontiers of Computer Science*, 18(6):186345.

Wu, Q., Bansal, G., Zhang, J., Wu, Y., Li, B., Zhu, E., Jiang, L., Zhang, X., Zhang, S., Liu, J., Awadallah, A. H., White, R. W., Burger, D., and Wang, C. (2023). Autogen: Enabling next-gen llm applications via multi-agent conversation. *arXiv preprint arXiv:2308.08155*.

Yajam, H., Ebadi, E., and Akhaee, M. A. (2024). Jabs: A blockchain simulator for researching consensus algorithms. *IEEE Transactions on Network Science and Engineering*, 11(1):3–13.

Yang, C., Wang, X., Lu, Y., Liu, H., Le, Q. V., Zhou, D., and Chen, X. (2024a). Large language models as optimizers. In *Proceedings of the 12th International Conference on Learning Representations*. ICLR 2024.

Yang, J., Jimenez, C. E., Wettig, A., Lieret, K., Yao, S., Narasimhan, K., and Press, O. (2024b). SWE-agent: Agent-computer interfaces enable automated software engineering. In *Advances in Neural Information Processing Systems*, volume 37.

Yang, S., Nayak, K., and Zhang, F. (2025). Decentralization of ethereum's builder market. In *2025 IEEE Symposium on Security and Privacy (SP)*, pages 1512–1530. IEEE.

Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., and Cao, Y. (2023). React: Synergizing reasoning and acting in language models. In *Proceedings of the 11th International Conference on Learning Representations*. ICLR 2023.

Zargham, M. and Rice, C. (2019). Introducing complex adaptive dynamics computer-aided design (cadcad). BlockScience blog. Accessed 4 Feb 2026.

Zhang, J., Gao, J., Li, Y., Chen, Z., Guan, Z., and Chen, Z. (2022). Xscope: Hunting for cross-chain bridge attacks. In *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*, pages 1–4, New York, NY, USA. Association for Computing Machinery.

Zhang, Q. and Li, H. (2007). MOEA/D: A multiobjective evolutionary algorithm based on decomposition. *IEEE Transactions on Evolutionary Computation*, 11(6):712–731.

A Literature Review

A.1 Blockchain

Blockchain is a distributed ledger, where network participants use a consensus protocol to agree on valid transactions. Bitcoin provided an early and influential application of this design (Böhme et al., 2015), and the technology has since been applied to systems that require shared records, traceability, or programmable execution, including a broad range of IoT settings (Khan et al., 2024). These applications also introduce new failure modes, particularly when assets and messages move across chains. Because live stress tests can expose networks and users to financial and operational risk, simulation provides a controlled setting for evaluating blockchain behavior under specified workloads and attacks.

Representative Applications. Blockchain has been studied in settings where several organizations need to maintain a shared record. In supply chain management, where a central concern is the tracking and coordination of goods from manufacturers to end customers (Tan, 2002), such records provide a shared and auditable view across organizations. Examples include *IBM Food Trust* (IBM, 2018; Kawaguchi, 2019) and the *Everledger* platform (CSIRO, 2022; Kshetri, 2022). In healthcare, *MedRec* uses blockchain-based permission management for access to medical records (Azaria et al., 2016). Energy applications use similar mechanisms for market coordination and auditable exchange (Teng et al., 2021); the *TRANSAX* platform, for example, supports transactions within decentralized microgrids (Laszka et al., 2018). Decentralized finance provides another prominent application class: Uniswap implements decentralized trading through an automated market maker (Adams et al., 2021), while Hyperliquid provides fully on-chain spot and perpetual-futures order books (Hyperliquid, 2026a;b). Such applications incorporate protocol rules with transaction demand, liquidity, and ordering, with which we may simulate and study congestion and strategic interactions.

Cross-chain Interoperability and Security Incidents. Cross-chain protocols transfer assets or messages between otherwise separate networks. They are widely used in decentralized finance, but the bridge logic required for these transfers adds key-management, validation, and authorization risks. Broad surveys classify DeFi vulnerabilities and their associated attack patterns (Li et al., 2022), and bridge-specific systematizations organize bridge components, exploit classes, and historical incidents (Lee et al., 2023; Belenkov et al., 2025). Automated tools such as *Xscope* inspect cross-chain transactions for signs of exploitation (Zhang et al., 2022). Individual incident reports illustrate the recurring failure classes: fraudulent deposit-address registration after a key exposure (Bifrost, 2022), privileged-invocation and verification bypasses (Rekt, 2021; 2022b), and missing recipient-contract validation (QANplatform, 2022). Related key-management and authorization failures also occur outside bridge logic, as in the Ankr staking-token exploit (Rekt, 2022a).

Experimental Methodology and Benchmarks. Blockchain measurements depend on the workload, network configuration, and selected performance metrics, so experimental comparisons require an explicit benchmark design. Gervais et al. (2016) use Markov decision processes to study double-spending and selfish-mining behavior in proof-of-work blockchains. For permissioned systems, *BLOCKBENCH* combines end-to-end workloads with layer-specific microbenchmarks to identify bottlenecks across the blockchain software stack (Dinh et al., 2017). *Gromit* instead uses a black-box transaction-fabric model and varies validator counts and network delays to compare scalability, throughput, latency, and sensitivity to network conditions across blockchain architectures (Nasrulin et al., 2022).

Blockchain Simulations for Protocol Design. Simulation allows researchers to test protocol changes and adversarial conditions without deploying them on a live blockchain. Early studies modeled network-layer attacks against Bitcoin (Neudecker et al., 2015), and evaluated selfish-mining strategies (Göbel et al., 2016). *BlockSim* provides configurable models of proof-of-work consensus and related blockchain processes (Alharby and van Moorsel, 2020). Other simulators focus on different protocol questions. *BlockEmulator* provides modular support for deploying and testing blockchain sharding protocols and their cross-shard transaction mechanisms (Huang et al., 2025), while *JABS* offers a modular environment for designing and evaluating consensus algorithms, including non-proof-of-work protocols (Yajam et al., 2024).

A.2 LLM Agents

LLM agents extend language models with mechanisms for planning, tool use, and memory. Surveys organize these systems around profile, memory, planning, and action components (Wang et al.,

2024), and planning-focused surveys further distinguish task decomposition, plan selection, external modules, reflection, and memory (Huang et al., 2024). These capabilities are relevant to digital-twin construction because the model must translate a system description into executable artifacts, invoke simulation tools, and revise its proposal after receiving validation or performance results.

Tool Use and API Grounding. Tool use connects an LLM to simulators, data sources, and executable validation routines. *ReAct* interleaves reasoning traces with external actions (Yao et al., 2023). *Toolformer* and *ToolLLM* study how models can learn or be instructed to invoke APIs (Schick et al., 2023; Qin et al., 2023). *Gorilla* further shows that retrieving API documentation at inference time can reduce incorrect tool calls when interfaces change (Patil et al., 2024). In our setting, these results support exposing simulators and blockchain RPC endpoints as tools with explicit input and output contracts.

Multi-Agent Decomposition. Multi-agent systems divide a workflow among model instances with different responsibilities. *AutoGen* and *CAMEL* use role definitions and structured exchanges to coordinate these instances (Wu et al., 2023; Li et al., 2023). *ChatDev* applies a similar design to software development by assigning separate roles to design, implementation, and testing (Qian et al., 2024). Spec2Twin-Chain uses distinct worker roles for experiment design, execution, and verification so that each output can be checked before it enters the next stage.

Feedback, Memory, and Iteration. Iterative workflows also need a way to reuse earlier results without retraining the model. *Reflexion* stores textual feedback in episodic memory for use in later attempts (Shinn et al., 2023). *Voyager* accumulates reusable programs and revises them after execution failures (Wang et al., 2023), while *Generative Agents* organizes observations into a memory stream that supports later planning (Park et al., 2023). These studies motivate retaining evaluated designs, objective values, and diagnostics as context for subsequent digital-twin proposals.

Interfaces and Guardrails. The interface between an agent and its tools affects what the agent can execute and how its actions are checked. *SWE-agent* reports that changes to repository navigation and test-execution interfaces improve performance on software engineering tasks (Yang et al., 2024b). Work on tool-use alignment also studies when a model should refuse harmful instructions, distrust insecure tool responses, and avoid tool calls it can answer directly (Chen et al., 2024). We therefore restrict worker actions through schemas, role-specific interfaces, and executable guardrails rather than relying on unconstrained model output.

The two bodies of work address complementary parts of our problem. Blockchain research supplies simulators, benchmark methods, and controlled adversarial scenarios. LLM-agent research supplies mechanisms for tool use, role separation, and feedback-guided iteration. Spec2Twin-Chain combines these elements for the narrower task of constructing and calibrating blockchain digital twins from an explicit specification and behavioral evidence.

B Optimization Strategy: Details

B.1 Upper-Level Optimization: LLM-Driven Architectural Search

The upper-level problem selects a discrete configuration $d \in \mathcal{D}$ that yields a high-quality optimized frontier $\mathcal{P}(\mathbf{J}_R(d, \theta^*(d)))$. This problem is difficult because the mapping from symbolic architecture to optimized performance is non-smooth. Moreover, the validity constraint $\mathbb{I}_{\text{ct}}(d) = 1$ can involve semantic and structural rules that are awkward to encode as ordinary integer constraints, but natural to express in a schema with language-level requirements.

LLMs are used as structured proposal mechanisms for this symbolic search. Their role is not to certify optimality. Rather, they provide informed candidate architectures, revise proposals under validation feedback, and use prior evaluated configurations to guide subsequent proposals. Each candidate remains subject to structural validation before simulation. Thus, the LLM functions as a proposal prior and interface layer, not as a substitute for formal validation or numerical optimization.

In this sense, (6) can be viewed as an instance of simulation-driven black-box optimization: candidate structures are generated without access to gradients, evaluated through an external compiler and simulation runtime, and subsequent proposals condition on previously evaluated outcomes (Shahriari et al., 2016; Snoek et al., 2012). Recent work that recasts black-box optimization as generative or conditional sampling includes inverse generative models (Kumar and Levine, 2020) and diffusion-based optimization (Krishnamoorthy et al., 2023; Li et al., 2024). Spec2Twin-Chain is also closely related to LLM-based optimization methods such as OPRO, in which an LLM generates new candidate solutions conditioned on earlier solutions and their objective values (Yang et al., 2024a).

But unlike these previous works, our Spec2Twin-Chain maintains a clear bi-level strategy that optimizes a separate lower-level problem from the upper-level loop. Here, we first formalize the upper-level loop as follows:

1. **Context initialization.** Construct a prompt context containing the construction request R, the target-system specification $\mathfrak{S}$, behavioral evidence E, the declarative schema, and the current objective contract $\mathbf{J}_R$. Similar to Retrieval-Augmented Generation (RAG), external task-specific information in our loop is supplied to the LLM at inference time, but here we perform deterministic and schema-aware retrieval rather than an unconstrained semantic document search.

2. **Candidate generation.** At iteration t, the LLM proposes a set of N candidate architectures

$$\{d_t^{(1)}, \ldots, d_t^{(N)}\} \sim P_{\text{LLM}}(d \mid \mathfrak{S}, E, \mathbf{J}_R, H_{t-1}), \tag{B.1}$$

   where H_{t-1} is the feedback context constructed from previously evaluated candidates.

3. **Schema validation.** Each candidate $d_t^{(i)}$ is checked against $\mathbb{I}_{\text{ct}}$. Invalid candidates are discarded or returned to the LLM with validator diagnostics for correction. In our Part 2.1 and 2.2 experiments, all 86 upper-level proposals were processed without invoking deterministic fallbacks. In addition, Part 2.2 contains 384 schema-validated worker decisions with no fallback response. But compile-repair turns are internal to candidates, this zero-fallback statistic should not be interpreted as a zero initial schema-rejection rate.

4. **Performance feedback.** Valid candidates are passed to the lower-level solver, which computes their optimized frontier or frontier summary. These outcomes are appended to the search history.

5. **In-context refinement.** The history H_t is formatted as a learning trajectory for the next iteration, allowing the LLM to condition future proposals on observed trade-offs among fit, stability, cost, and feasibility.

B.2 Lower-Level Optimization: Simulation-Based Parameter Refinement

Conditioned on a valid architecture d, the lower-level problem becomes a simulation-based optimization problem over $\Theta(d)$. The framework does not require a unique lower-level optimizer: derivative-free methods such as evolutionary algorithms, decomposition-based multi-objective search, surrogate-assisted Bayesian optimization, or scalarized optimizers may be appropriate depending on the objective structure and evaluation budget (Zhang and Li, 2007; Snoek et al., 2012;

Forrester et al., 2007; Hansen, 2016). In this study, we instantiate the lower level with the Non-dominated Sorting Genetic Algorithm II (NSGA-II) because it maintains a population approximation to the Pareto frontier, preserves diversity through crowding distance, does not require gradients, and naturally accommodates noisy and constrained simulation outputs (Deb et al., 2002). This choice is therefore a practical default for the multi-objective, simulation-based setting, not a restriction of the Spec2Twin-Chain formulation. Appendix E provides the methodological complement, including the feasibility-aware solver loop in Appendix E.1, and the executable interpretation of the formulation in Appendix E.2.

Let $\mathcal{Q}_g$ denote the population of tuning vectors at generation g. For each $\theta_i \in \mathcal{Q}_g$, the simulator evaluates the configuration (d, θ_i) and returns the objective vector $\mathbf{J}_R(d, \theta_i)$ together with guardrail violations. Ranking follows a feasibility-aware dominance principle: feasible solutions dominate infeasible ones; infeasible solutions are ordered by their degree of constraint violation; and among feasible solutions, Pareto dominance and crowding distance guide selection. Under a finite evaluation budget, the resulting procedure returns an empirical approximation to the architecture-conditioned Pareto frontier, namely the non-dominated feasible set (Deb et al., 2002) among the evaluated points in $\Theta(d)$. Algorithm 2 in Appendix E.1 states the full loop.

B.3 Global Archive and Feedback Construction

The bi-level framework does not treat lower-level searches as isolated runs. Instead, it maintains a global Pareto archive $\mathcal{A}_{\text{global}}$ across upper-level iterations. After computing the architecture-conditioned empirical Pareto frontier $\mathcal{P}_d^*$ for a candidate, the archive is updated by non-dominated merge:

$$\begin{aligned} \mathcal{U}_d &= \mathcal{A}_{\text{global}} \cup \mathcal{P}_d^*, \\ \mathcal{A}_{\text{global}} &\leftarrow \{c \in \mathcal{U}_d : \nexists c' \in \mathcal{U}_d,\ c' \prec c\}. \end{aligned} \tag{B.2}$$

where $\prec$ denotes Pareto dominance under the request-conditioned objectives and guardrail treatment.

The archive also serves as semantic memory for the upper-level architect. To construct the feedback context H_t, a representative subset $\mathcal{E}_t \subset \mathcal{A}_{\text{global}}$ is selected. When scalar ranking is required, we use a request-conditioned utility

$$U_R(d, \theta) = w^\top \widetilde{\mathbf{J}}_R(d, \theta), \tag{B.3}$$

where $\widetilde{\mathbf{J}}_R$ is obtained by first orienting every component-wise objective to a maximization problem: we keep $z_j = J_j^R$ for maximization tasks and negate $z_j = -J_j^R$ for minimization tasks. Then we apply within-archive min–max normalization: $\widetilde{J}_j^R = (z_j - \min z_j)/(\max z_j - \min z_j)$. The selected exemplars are summarized into a compact feedback context that connects design choices to realized simulation performance. This feedback construction implements in-context learning: the model receives evaluated examples from the prior search trajectory and adapts its next proposal distribution without weight updates, consistent with few-shot prompting and broader surveys of in-context learning (Brown et al., 2020; Dong et al., 2023). We include both successful and diagnostic information (objectives, guardrail violations, and structural edits) because demonstration selection and formatting can materially affect in-context learning behavior (Min et al., 2022). The archive is therefore a validated record of the search trajectory, not an unconstrained conversational log. In this way, the LLM's subsequent proposals are grounded in evaluated design trajectories. Algorithm 1 summarizes the complete loop.

B.4 Post-Construction Optimization and Reuse

Once a calibrated twin is available, the same hierarchical search machinery can also be reused in a distinct *post-construction* stage, where the digital twin is further optimized for a downstream decision problem. This is analogous to an adaptation stage at the workflow level but different from an LLM post-training in that the model weights are not updated. The same bi-level structure can also be used after the initial twin has been constructed. In downstream instances, the optimizer conditions on a task-specific external input X_e, which may be a calibrated digital twin, a controlled simulator, historical blockchain traces, protocol specifications, benchmark scenarios, or a blockchain system exposed by observation, replay, or query. The downstream problem is then defined over a new architecture or policy structure d_e and associated tuning variables θ_e:

$$\min_{d_e \in \mathcal{D}_e,\ \theta_e \in \Theta_e(d_e; X_e)} \mathbf{J}_e(d_e, \theta_e; X_e), \tag{B.4}$$

with experiment-specific structural-validity and guardrail constraints. Under this view, a calibrated digital twin is one possible substrate rather than a required input: if X_e is a built twin, the task is DT-enabled optimization; if X_e is direct blockchain evidence, benchmark traces, or an observable blockchain interface, the same hierarchical search pattern supports evidence-conditioned blockchain optimization unless the task explicitly constructs a new twin. The lower-level solver is selected for the downstream task and need not be NSGA-II. It may use another simulation-optimization method suited to the objective structure, parameter domain, and evaluation budget, provided that it optimizes θ_e conditional on d_e and enforces the declared guardrails.

C SUPPLEMENTARY EXPERIMENTS ON PART 2

Table 3: Experimental instances under the Spec2Twin-Chain bi-level formulation.

Experiment	Substrate and purpose	Bi-level optimization contract
Part 1 BlockSim calibration	Ethereum proof-of-work mempool twin calibrated to BlockSim KPI evidence; tests the core construction capability of the builder.	d**:** mempool, propagation, consensus, fee-market, and block-production modules. θ**:** timing, rate, gas, capacity, mining, and latency variables. **Objective:** minimize KPI fit error with stability and cost terms.
Part 2a Congestion	Congestion-regime scenario over the calibrated mempool twin; tests whether the twin can expose queue, latency, and breakdown behavior.	d**:** load-profile, queue, backpressure, bottleneck, and telemetry structure. θ**:** traffic multiplier, queue threshold, backpressure coefficient, elasticity exponent, and latency sensitivity. **Objective:** match throughput behavior while monitoring latency, queue stress, and guardrails.
Part 2b Adversarial stress	Joint mempool–adversary stress twin over controlled adversarial scenarios; tests aggregate stress matching and class-fingerprint recovery.	d**:** attack channel, adversary type, observation surface, fingerprint logic, and detection structure. θ**:** attack intensity, timing, concealment, disruption, and detection-threshold variables. **Objective:** match attack profiles while balancing disruption, detectability, stability, and cost.
Part 2.1a MEV-style block construction	Validator-side block-building policy over the mempool substrate; tests controlled reward capture under a 12-second slot budget.	d**:** bundle selection, conflict resolution, ordering, pruning, timeout, and fallback policy structure. θ**:** search budgets, pruning thresholds, scoring weights, conflict penalties, and timing controls. **Objective:** maximize reward subject to valid-block, conflict, and slot-time constraints.
Part 2.1b Inclusion policy	Sender-side fee and replacement policy over the mempool substrate; tests reliability–cost trade-offs under controlled inclusion scenarios.	d**:** adaptive fee rule, replacement strategy, urgency tiers, and target-horizon policy. θ**:** fee multiplier, fee cap, replacement bump, retry interval, target horizon, and urgency threshold. **Objective:** achieve target inclusion probability while controlling fee cost, overpayment, and invalid replacements.
Part 2.2 GSMP-style update policy	Update-timing supervision over a dynamic extension of the controlled Part 2.2b inclusion-policy environment, initialized from the reported sender policy and anchored to the Part 1 calibration evidence.	d**:** GSMP-style update-scope structure: review clocks, validity bands, and keep, fast-update, or full-update actions. θ**:** the six Part 2.2b sender-policy variables, grouped into a four-variable fast set and the full six-variable set. **Objective:** minimize an operational inclusion, fee, and stability score on locked held-out paths, reporting update activity separately.

C.1 PART 2.1: POLICY OPTIMIZATION WITH THE CALIBRATED MEMPOOL TWIN

Part 2.1 is conceptually different from a digital-twin construction task. Here the calibrated mempool digital twin is fixed as a decision substrate, while the object to be optimized is a downstream policy. We therefore interpret Part 2.1 as an optimization task facilitated by one digital twin rather than as a construction of another. Admittedly, these downstream problems could be solved by other black-box optimizers, but we retain our bi-level workflow to demonstrate that the same machinery can be modified and executed even after a twin has already been established. Part 2.2a examines validator-side MEV-style block construction under a 12-second slot-time budget, and Part 2.2b examines sender-side transaction-inclusion fee policy. The question is whether the bi-level workflow can construct a policy structure d and tune the associated policy variables θ to improve a controlled utility objective while preserving feasibility and guardrails. Table 4 summarizes the results; Appendix D.4 provides the detailed MEV-style reward and slot-time visualization in Figure 8 and the inclusion-policy baseline comparison in Table 12.

Table 4: Part 2.1 policy-optimization summary over the calibrated mempool twin.

Study	Proposals	J_{fit}	J_{stab}	J_{cost}	Main feasible outcome
Part 2.2a: MEV-style block construction	24	0.0000	0.0000	4200.0000	Full reward capture relative to the controlled reference; included bundles `alpha` and `beta`; reward gain of 1.6575 ETH; valid block in 4.2 seconds under a 12-second budget.
Part 2.2b: Inclusion policy	24	0.0066	0.0053	159.8878	Minimum inclusion probability of 0.8016; total fee of 159.3803 gwei; overpayment of 0.5075 gwei; no invalid replacements.

C.1.1 Part 2.1a: MEV-Style Block Construction Under Slot-Time Budget

Part 2.2a uses a controlled block-building scenario with public and private bundle candidates, archived reward values, latency terms, risk discounts, and conflict metadata. The policy family is a conflict-aware exact-subset solver: it indexes conflicts, scores adjusted value, searches feasible bundle subsets, applies deterministic tie-breaking, and enforces timeout control. The lower level tunes search-depth, candidate-limit, risk-cap, latency-penalty, surrogate-fee, and timeout parameters.

In this experiment, J_{fit} has a task-specific interpretation that differs from the behavioral calibration discrepancy used in Part 1. Here,

$$J_{\text{fit}} = |\Delta R - \Delta R_{\text{ref}}|\,,$$

where ΔR denotes the reward improvement achieved by the candidate policy over the baseline and ΔR_{ref} denotes the reward improvement of the controlled reference outcome. Thus, $J_{\text{fit}} = 0$ in Part 2.2a means that the optimized policy matches the reference reward improvement. It does not indicate zero behavioral calibration error.

The optimized policy includes `bundle_alpha` and `bundle_beta`, rejects conflicting alternatives, and produces a valid block in 4.2 seconds under the 12-second budget. Captured value is 4.0575 ETH, compared with 2.4000 ETH for the controlled reference, giving a reward delta of 1.6575 ETH. The result shows that once the policy family is expressed as conflict-aware exact subset search, lower-level tuning can recover the scenario optimum without sacrificing valid-block or slot-time constraints. The corresponding reward and slot-time comparison is visualized in Appendix D.4, Figure 8.

For Part 2.2a in particular, the result is not a live MEV-Boost benchmark. A real comparison against MEV-Boost or production block builders would require historical slot-level data, mempool and private-orderflow observations, relay bids, builder payloads, realized proposer values, and a same-information evaluation protocol over many slots. It would also require high-throughput EVM/state simulation and latency-aware bundle validation under proposer-time constraints. The present controlled scenario is therefore used to evaluate DT-enabled policy construction, not to claim superiority over live MEV-Boost.

C.1.2 Part 2.1b: Transaction-Inclusion Fee Policy Optimization

Part 2.2b evaluates sender-side fee and replacement behavior across low-congestion, high-congestion, fee-spike, replacement, and target-horizon scenarios. The upper level defines an adaptive-urgency policy with a calibrated fee oracle, urgency gate, bounded replacement manager, retry scheduler, cap enforcer, and telemetry loop. The lower level tunes the base-fee multiplier, fee cap, replacement bump, retry interval, target horizon, and urgency threshold.

Figure 4 exposes the lower-level optimization result behind the selected policy. Each circle is a unique objective triple from the feasible run-scoped NSGA-II archive for the fixed Codex-authored policy structure, with marker size reflecting repeated archive entries at the same objective values. The archive records 29 feasible entries, corresponding to 13 unique objective triples, after 288 lower-level evaluations and five completed generations. The reported policy is the lowest-cost feasible archive member: it clears the minimum inclusion-probability target with a worst-case inclusion probability of 0.8016, pays 159.3803 gwei, and incurs only 0.5075 gwei of overpayment. The full baseline comparison and tuned-policy summary are reported in Appendix D.4, Table 12. In this controlled setting, the result shows how a calibrated digital twin can support policy comparison.

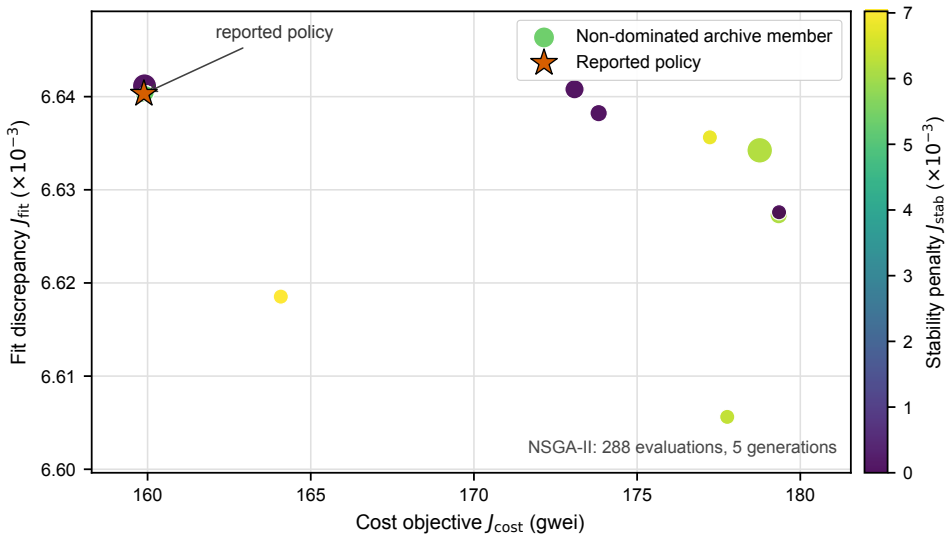


Figure 4: Lower-level optimization frontier for the Part 2.2b sender-side inclusion-policy study. Each circle is a unique objective triple in the feasible run-scoped NSGA-II archive for the Codex-authored policy structure; marker size reflects repeated archive entries at the same objective values. The horizontal axis reports the cost objective J_{cost}, the vertical axis reports fit discrepancy J_{fit}, and color reports the stability penalty J_{stab}. The highlighted solution is the policy reported in Table 4.

C.2 Part 2.2: GSMP-Style Update Timing for the Optimized Inclusion Policy

Parts 1 and 2.2 provide a calibrated mempool setting and a tuned sender-side inclusion policy. Part 2.3 asks what should happen to that policy afterwards. As transaction, block, and fee observations arrive, re-optimizing the deployed policy at every observation can chase transient noise and destabilize behavior that was already acceptable. The study therefore treats update timing itself as the decision problem. It extends the controlled Part 2.2b inclusion-policy environment along locked dynamic paths, initializes every arm from the reported sender policy, and uses the Part 1 results as calibration evidence for the broader mempool setting.

We model the update decision through a GSMP-style event-clock representation of the policy's operating environment (Glasserman and Yao, 1992); the correspondence to the GSMP formalism is given in Appendix D.5. The discrete scheme consists of transaction, block, inclusion, replacement, and review events. Its clock state records hourly and daily reviews, persistence outside the validity band, retry timing, and time since the last accepted update. At each review the supervisor also observes rolling base-fee and mempool-pressure estimates and recent inclusion, fee, and replacement outcomes. The daily review supersedes the hourly one when the two coincide. The action set is deliberately small: keep the current parameters, apply a fast update that re-optimizes the four fast policy variables, or apply a full update of all six variables, with full updates available only at daily reviews. Four update policies are compared under this contract: never update, always update, a rule-based selective baseline whose state-to-action mapping was tuned on development seeds and then frozen, and LLM-guided selective updating, in which a live worker chooses the action at each review. All four start from the same reported Part 2.2b parameter vector, observe the same event paths under common random numbers, and use the same lower-level optimizer, bounds, and per-invocation budget whenever re-optimization is requested. Every protocol choice was frozen before the locked evaluation, in which the worker made 384 review decisions, one at each review, without access to hidden regime labels or future events. Appendix D.5 records the full protocol.

The objective here is user-specified rather than an instance of the construction families in Section 2.1, so we give it its own symbol. The operational score J_{op} combines inclusion reliability at the target horizon, fee and overpayment, and the stability of realized inclusion performance. It deliberately excludes optimizer, worker, and update costs. An always-updating policy is therefore not charged

for the act of updating and can lose only through worse realized outcomes; update activity is reported separately.

Table 5: Part 2.2 locked update-policy comparison over eight held-out seeds. Lower J_{op} is better. Updates per day counts accepted parameter changes; optimizer launches, including refits that returned no accepted change, are reported in Appendix D.5.

Update policy	J_{op}	**Mean inclusion prob.**	**Updates/day**
Never update	0.0974	0.9370	0.0000
Rule-based selective	0.0950	0.9440	1.4375
Always update	0.0996	0.9342	15.1875
LLM-guided selective	0.0924	0.9411	1.0625

Table 5 reports the locked comparison over eight held-out seeds. LLM-guided selective updating attains the best operational score with about one accepted update per day. The paired differences against each baseline favor the LLM policy on all eight seeds, and their 95% confidence intervals exclude zero (Appendix D.5, Table 13). Figure 5 shows the mechanism on one locked seed. The worker keeps the deployed parameters through three short fee shocks, requests one targeted fast update after mempool conditions have remained outside the validated band, and applies one retune after conditions return to the band. Never updating avoids churn but concedes inclusion reliability once conditions shift persistently. Always updating performs worst despite paying no update penalty. Its repeated refits use short trailing windows, and the resulting pattern is consistent with overreaction to transient spikes.

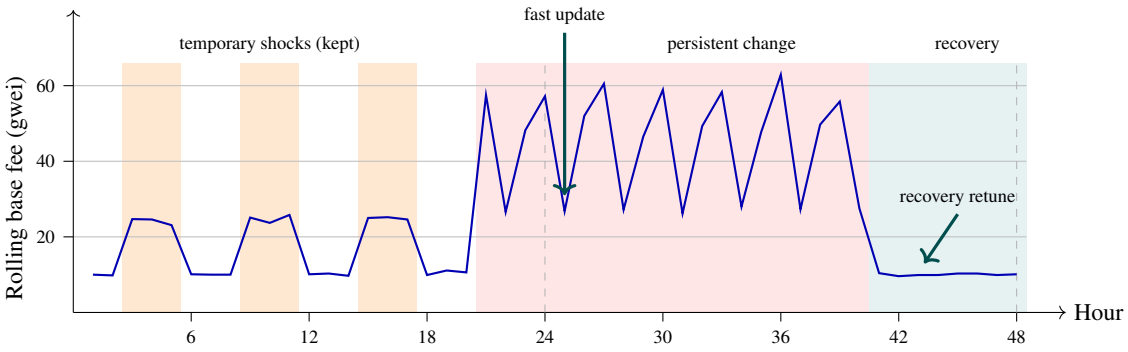


Figure 5: Part 2.2 review trace for one locked seed under LLM-guided selective updating. The line is the rolling base-fee estimate observed at each review; shaded bands are the hidden scenario periods, used only for post-run classification and never shown to the worker; dashed vertical lines mark daily reviews. The worker keeps the deployed parameters through three temporary fee shocks, accepts one targeted fast update once mempool conditions have remained outside the validated band, and accepts one retune after conditions return to the band.

The claim is scoped to this controlled setting: a dynamic extension of the Part 2.2b inclusion-policy evaluator, a finite update-policy set, and locked synthetic event paths. The evidence supports selective, evidence-conditioned update timing, and it shows a measurable contribution from the live worker beyond the frozen rule-based supervisor. It does not establish a universal update cadence and does not validate a live Ethereum fee policy.

Taken together, Part 2 shows how calibration evidence can support controlled stress analysis, policy optimization, and online policy-maintenance tasks. The downstream routines screen candidates against declared guardrails, expose diagnostic limitations, and produce interpretable decision outputs. These experimental claims remain scoped to controlled simulators and curated benchmark scenarios, which is appropriate before live-system validation.

D Experiment Details and Additional Results

D.1 Experimental Runtime and Reproducibility Details

This appendix records the run settings that are summarized in Section 3. Proposal counts refer to accepted Codex-authored upper-level candidates evaluated by the methodology runtime. Fallbacks denote deterministic candidate designs injected by the runtime when an LLM-generated proposal is unavailable or invalid. In the reported Part 2 runs, no fallback candidate is used. For Part 2.3, the proposal column instead counts bounded per-review update-scope decisions by the live worker; each is a schema-validated worker call that selects an update action, not a candidate architecture.

Table 6: Runtime settings for the reported experiment instances.

Experiment	Proposals	Fallbacks	Lower-level setting	Termination / budget note
Part 1 calibration	–	–	Population 160; 80 optimizer generations; Stage-2 lower-level budget of 20 iterations	Up to three upper-level construction iterations under full BlockSim evidence.
Part 2.1a congestion	32	0	Mixed-variable NSGA-II; population 320; maximum 240 generations	Plateau after 9,920 evaluations and 30 completed generations.
Part 2.1b adversarial stress	6	0	Mixed-variable NSGA-II; population 160; maximum 100 generations	Plateau after 3,360 evaluations and 20 generations.
Part 2.2a MEV-style block construction	24	0	Mixed-variable NSGA-II; population 64	Plateau after 1,344 evaluations and 20 generations.
Part 2.2b inclusion policy	24	0	Mixed-variable NSGA-II; population 48	Plateau after 288 evaluations and five generations.
Part 2.3 update policy	384	0	Seeded bounded random search; 64 candidates per launch; trailing three-hour calibration window	Fixed 48-review paths on eight locked seeds; feasible improvement required; no plateau rule.

D.2 Part 1 Calibration and Feedback Details

Table 7 reports the full KPI comparison for the Part 1 calibration experiment. The aggregate fit error is the mean relative discrepancy over the ten comparable KPIs. The timing metrics account for most of the residual error, while count, throughput, gas, and uncle metrics are matched exactly or nearly exactly.

Table 7: Part 1 full-evidence calibration run: KPI comparison between BlockSim evidence and the generated digital twin.

KPI	BlockSim	Digital twin	Relative error
Blocks	33	33	0.0000
Total transactions	4534.0000	4534.0000	0.0000
Transactions per block	137.3939	137.3939	0.0000
Average block time	15.2211	13.0683	0.1414
Total uncle count	9.0000	9.0000	0.0000
Uncle rate	0.2727	0.2727	0.0000
Total gas used	199106942.0000	199106076.0000	0.000004
Throughput (TPS)	9.3086	9.3086	0.000002
Block time, 95th percentile	33.9403	32.4357	0.0443
Block time, 99th percentile	35.3770	36.7589	0.0391
Aggregate fit error			0.022484
Target threshold			0.050000

The feedback studies weaken the initial request by removing evidence-guided parameter bounds. Figure 6 and Table 8 report the resulting recovery behavior. With full diagnostic feedback, the loop reaches the calibrated fit level by the fourth iteration. With progressive feedback, recovery is slower, but the final fit error still falls below the target threshold.

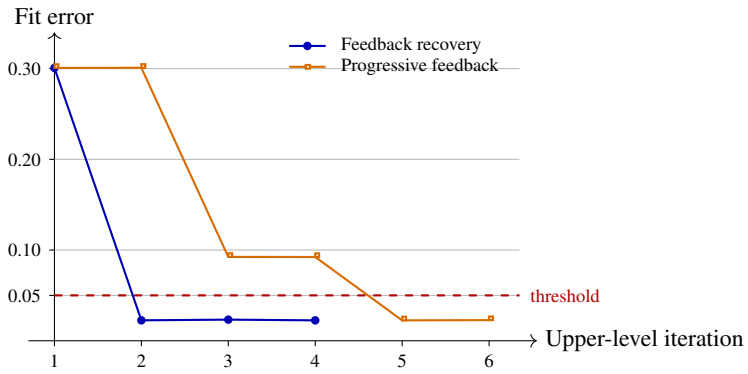


Figure 6: Feedback recovery behavior under weakened request conditions. When evidence-guided bounds are removed, the initial candidate is poorly calibrated. Diagnostic feedback allows the bi-level loop to recover below the target fit-error threshold of 0.05. The progressive-feedback setting recovers more slowly because detailed diagnostic information is deliberately delayed.

Table 8: Part 1 feedback ablations. Both ablations begin from weaker request conditions than the full-evidence calibration run.

Study	Condition	Initial fit error	Best fit error
Full-evidence calibration run	Full BlockSim evidence with evidence-guided parameter bounds	–	0.022484
Feedback-recovery ablation	Weakened request without evidence-guided bounds; full diagnostics after first iteration	0.300786	0.022484
Progressive-feedback ablation	Weakened request without evidence-guided bounds; diagnostic feedback revealed gradually	0.300786	0.022486

D.3 Part 2.1 Stress-Regime Details

Table 9 reports the objective and runtime metrics for the congestion-regime experiment. The optimized congestion parameters are a traffic multiplier of 1.0000, queue threshold of 1750.522 transactions, backpressure coefficient of 0.6244, elasticity exponent of 1.6862, and latency sensitivity of 0.1511.

Table 9: Part 2.1a congestion-regime run. Objective values and runtime metrics are reported for the Codex-authored architecture after lower-level optimization.

Quantity	Value
Codex-authored upper-level proposals	32
Worker fallbacks	0
Lower-level evaluations used	9,920
J_{fit}	0.0000
J_{stab}	1254.4253
J_{cost}	84.4565
Conservative breakdown threshold	76 TPS
RMS throughput error	0.0000
Stress-point throughput	42.9630 TPS
Stress-point latency	258.8642 ms
Stress-point queue depth	292.4095 txs
Eviction rate	0.0000
Diverged	No
Guardrails satisfied	Yes

Table 10 reports the scenario-level thresholds. The conservative threshold used in the main text is the minimum threshold across the three controlled stress scenarios.

Table 10: Scenario-level congestion thresholds for Part 2.1a.

Stress scenario	Threshold (TPS)	Min throughput (TPS)	Max queue (txs)	Peak latency (ms)
Peak-load TPS spike	92.0	48.0000	83.3585	145.0698
Gas-spike trace	76.0	43.8710	80.7231	232.6230
Throughput-collapse trace	88.0	42.9630	292.4095	258.8642

Table 11 and Figure 7 report the adversarial-stress diagnostics. The key distinction is between aggregate profile fit and strict class-fingerprint recovery: the aggregate profile fits exactly, but only two of five strict scenario fingerprints match. At the class level, the mempool-spam and frontrunning-injection scenarios satisfy the strict fingerprint oracle, whereas the eclipse-style, selfish-timing, and covert-replacement scenarios remain below the strict class-signature criteria.

Table 11: Part 2.1b adversarial-stress run. The run covers five controlled attack classes and reports objective values after lower-level optimization.

Quantity	Value
Codex-authored upper-level proposals	6
Worker fallbacks	0
Lower-level evaluations used	3,360
Attack classes covered	5
Attack-frontier rows emitted	80
J_{fit}	0.0000
J_{stab}	0.3318
J_{cost}	169.1918
Profile-fit error	0.0000
Strict fingerprint match rate	0.4000
Mean detectability score	0.3384
Mean detection latency	6.6000 blocks
Guardrail violations	0
Guardrails satisfied	Yes

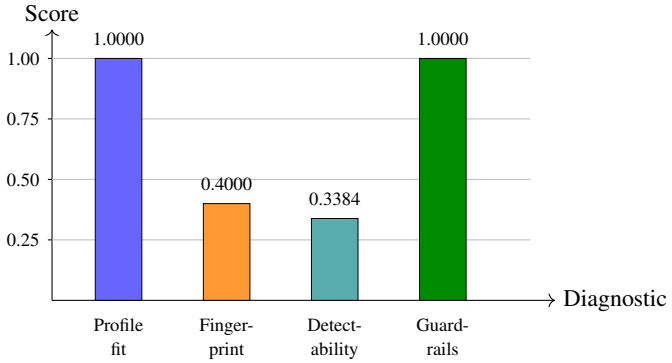


Figure 7: Part 2.1b adversarial-stress diagnostics. The run obtains exact aggregate profile fit and satisfies guardrails, but only two of the five strict class fingerprints match the scenario oracle. This partial fingerprint match is treated as a diagnostic limitation rather than hidden by the aggregate objective.

D.4 Part 2.1 Policy-Optimization Details

Figure 8 reports the reward and slot-time outcomes for the MEV-style block-construction scenario. The optimized conflict-aware exact-subset policy includes `bundle_alpha` and `bundle_beta`, excludes the conflicting alternatives, and remains inside the 12-second application budget. The tuned block-construction policy uses the largest allowed exact-subset and bundle-candidate limits in

the controlled scenario, assigns zero latency penalty for this deterministic slot instance, and reserves no additional timeout slack because the constructed block remains well inside the 12-second budget.

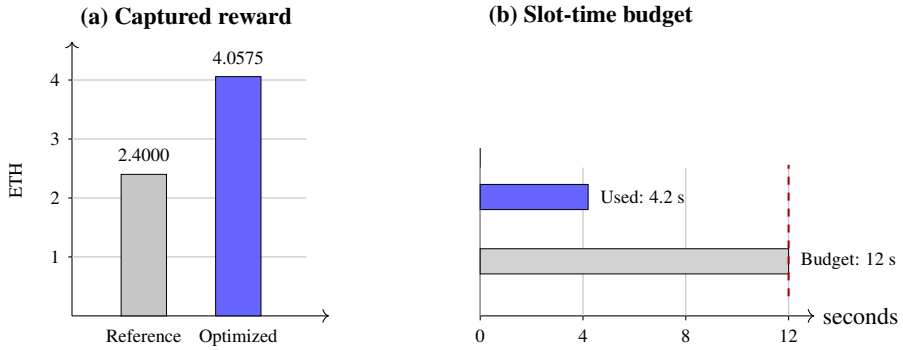


Figure 8: Part 2.1a MEV-style block-construction outcome. The optimized conflict-aware exact-subset policy captures the full reward improvement relative to the controlled reference while satisfying the valid-block and 12-second slot-time constraints.

Table 12 reports the inclusion-policy baselines. The optimized sender clears the target reliability threshold with substantially less fee expenditure than the conservative high-fee baseline. The optimized sender parameters are a base-fee multiplier of 1.2507, replacement bump of 18.3833%, retry interval of 50,475.8 ms, target horizon of 6.5920 blocks, and urgency threshold of 0.7533. These values implement a wait-under-normal-conditions policy that escalates only when inclusion reliability approaches the target boundary.

Table 12: Part 2.1b inclusion-policy comparison. The optimized policy is compared with naive low-fee and conservative high-fee baselines under the same controlled scenario set.

Policy	Min incl. prob.	Mean incl. prob.	Fee (gwei)	Overpay (gwei)	Target sat. rate
Naive low-fee sender	0.7404	0.8422	156.6336	0.0000	0.4000
Conservative high-fee sender	0.9553	0.9878	392.1200	109.0000	1.0000
Codex + NSGA-II optimized sender	0.8016	0.9087	159.3803	0.5075	1.0000

D.5 Part 2.2 Update-Policy Details

The update-timing study uses disjoint seed sets. Three development-tuning seeds were used to design the dynamic paths, observable state, worker prompt, parameter grouping, optimizer settings, and rule-based baseline. Three additional development-validation seeds were used to check the frozen protocol before the eight locked test seeds were opened. Locked outcomes were not fed back into the design. Each seed generates a 48-hour event path with hourly reviews and daily reviews at hours 24 and 48. The daily review supersedes the hourly one, so each policy faces 48 reviews per seed and the LLM arm makes 384 review decisions in total. Every decision is a separate schema-validated worker call with a recorded session; all 384 sessions are distinct and no fallback response was used.

The four fast variables are the base-fee multiplier, urgency threshold, retry interval, and replacement bump. The two slow variables are the fee cap and target horizon. Fast updates may be requested at any review and re-optimize only the fast group; full updates cover all six variables and are available only at daily reviews. A requested update uses seeded bounded random search over 64 candidates, including the currently deployed parameters, the reported Part 2.2b parameters, and frozen development warm starts. Candidate evaluation uses the trailing three-hour calibration window. A candidate is accepted only if it satisfies the review-time guardrails and improves the operational objective. The guardrails require an inclusion probability of at least the larger of 0.80 and the scenario-specific requirement on each calibration episode, compliance with the scenario fee cap and reported parameter bounds, and zero invalid replacements. If no improving feasible candidate is found, the deployed policy remains active and the failed attempt is recorded.

For episode t, let p_t be the realized inclusion probability at the target horizon, q_t the scenario-specific required probability, f_t the paid fee, c_t the fee cap, o_t the overpayment, and m_t the minimum sufficient fee. The frozen operational objective is

$$\begin{aligned} J_{\mathrm{op}} &= 0.55L_{\mathrm{rel}} + 0.30L_{\mathrm{fee}} + 0.15L_{\mathrm{stab}}, \\ L_{\mathrm{rel}} &= 0.70\,\frac{\overline{(0.90 - p_t)_+}}{0.90} + 0.30\,\overline{\mathbb{I}\{p_t < q_t\}}, \\ L_{\mathrm{fee}} &= 0.65\,\frac{\overline{f_t}}{c_t} + 0.35\,\overline{\frac{o_t}{\max(m_t, 1)}}, \\ L_{\mathrm{stab}} &= 0.50\ \mathrm{sd}_{\mathrm{pop}}(p_t) + 0.50\,\overline{|p_t - p_{t-1}|}. \end{aligned} \tag{D.1}$$

Here the overline denotes the average over the evaluated episodes, and $\mathrm{sd}_{\mathrm{pop}}$ the population standard deviation across them. The objective contains no optimizer, worker, launch, or update cost. The paired comparisons below use two-sided 95% Student-t intervals over the eight matched seed-level differences.

Table 13: Part 2.2 paired comparisons on the eight locked seeds. Differences are comparator minus LLM-guided selective updating in J_{op}, so positive values favor the LLM policy. Every per-seed difference is positive for all three comparators.

Comparator	Mean difference	95% CI
Never update	0.0050	[0.0028, 0.0072]
Always update	0.0072	[0.0039, 0.0106]
Rule-based selective	0.0026	[0.0009, 0.0042]

Table 14: Part 2.2 update activity and score components on the locked seeds. Optimizer launches count every re-optimization attempt; accepted updates count launches that changed the deployed parameters. The component values enter J_{op} through the normalization and weights frozen before the locked run.

Update policy	Launches/day	Accepted/day	Reliability	Fee	Stability
Never update	0.0000	0.0000	0.0144	0.2731	0.0504
Rule-based selective	1.6250	1.4375	0.0081	0.2808	0.0421
Always update	24.0000	15.1875	0.0218	0.2692	0.0458
LLM-guided selective	1.0625	1.0625	0.0068	0.2756	0.0399

The activity and component breakdown in Table 14 locates each failure mode. Never updating carries the largest stability component and the second-largest reliability component: the frozen parameters lose inclusion reliability once mempool conditions shift persistently, and its minimum inclusion probability (0.7747) and target-horizon satisfaction rate (0.9766) are among the weakest of the four policies. Always updating attempts a refit at all 24 reviews per day, accepts about 15 changes per day, and still records the worst reliability component and the lowest satisfaction rate (0.9479). The LLM policy accepts about one update per day, every launch it requests is accepted, and it attains the best reliability and stability components with a satisfaction rate of 0.9896. Its 384 decisions comprise 367 keep actions, 16 fast updates, and one full update. No update is accepted during a temporary-shock period. Every seed receives at least one targeted update during the persistent-change period, one seed receives a second fast update, and every seed receives one recovery retune after re-entry into the validated band.

The guardrails screen candidate updates on the trailing calibration window; they do not guarantee every subsequent stochastic outcome. On the locked paths, the LLM policy records zero invalid replacements and an aggregate minimum inclusion probability of 0.7994. The latter is a narrow miss of the 0.80 review-time threshold on one seed and is reflected in the reported satisfaction rate.

Connection to the GSMP formalism. A generalized semi-Markov scheme specifies states, a finite set of event types, a per-state list of active events, and a transition map; the scheme is driven by clock samples, and it can be read as a deterministic mapping from those samples to event epochs and counts (Glasserman and Yao, 1992). The Part 2.3 environment instantiates this structure, and

Table 15 states the correspondence. Two boundaries are deliberate. First, fixing the clock samples fixes the entire event path. This is the formal content of the common-random-number design: the four update policies face the same realized event stream and differ only in which update actions they accept at review epochs. Second, the classical formalism has no decision maker. The supervisor's choice among keep, fast update, and full update at each review is the controlled extension that this experiment adds, which is why the main text writes GSMP-style rather than claiming the classical model.

Table 15: Correspondence between the GSMP formalism and the Part 2.2 update environment.

GSMP element	**Realization in the Part 2.2 environment**
State	Deployed policy parameters and version; rolling base-fee and mempool-pressure estimates; validity-band status; pending transactions; persistence and cooldown counters
Event types	Transaction and block events; fee shocks; hidden regime changes; retry and replacement events; hourly and daily reviews
Active-event list	What can currently fire: a retry only while a transaction is pending; a fast update only at a review; a full update only at a daily review; the daily review supersedes the hourly one
Clock samples	The locked seeds' random draws (arrival timing, shock timing, regime sojourn durations) together with the deterministic review clocks
Event epochs and counts	The recorded review trace: per-review observables, accepted updates per day, and optimizer launches
Fixed clock samples fix the event path	The common-random-number comparison: identical event streams across all four policies, differing only in the accepted update actions

E Methodological Details and Executable Instantiation

This appendix complements the main methodology by recording the lower-level solvers used in the reported studies and the executable interpretation of the abstract formulation. Appendix E.1 gives the feasibility-aware NSGA-II procedure used in Part 1 and Parts 2.1–2.2; the Part 2.3 task-specific bounded search is described in Appendix D.5. Appendix E.2 clarifies how request-conditioned objectives, admissible tuning spaces, validation checks, and archive feedback are implemented without changing the mathematical formulation.

E.1 Lower-Level NSGA-II Solver

For a fixed architecture d, the lower-level problem is a constrained multi-objective simulation-optimization problem over the architecture-conditioned domain $\Theta(d)$. In the NSGA-II-based studies, $\Theta(d)$ is a mixed-variable tuning surface: it may contain bounded numeric variables and, when exposed by the specification, discrete or categorical choices. We use NSGA-II with feasibility-aware ranking. For each sampled tuning vector θ, the simulation oracle returns the objective vector $\mathbf{J}_R(d, \theta)$ and guardrail information. Feasible solutions dominate infeasible ones; among infeasible solutions, candidates with lower feasibility penalties are preferred; among feasible solutions, Pareto dominance and crowding distance guide selection. The standard evolutionary loop is summarized in Algorithm 2.

E.2 Executable Instantiation and Scope

The executable system instantiates the abstract formulation through request interpretation, structural validation, simulation evaluation, and archive-backed feedback. A construction request is normalized into an objective contract, admissible tuning surface, and guardrail set before lower-level optimization. Candidate architectures are evaluated only after structural checks; simulation outputs are converted into the request-conditioned objectives used by the optimizer. Engineering artifacts such as schemas, manifests, prompt templates, and reproducibility logs are implementation devices for enforcing Eq. (2.2); they are not additional mathematical assumptions.

The request-conditioned objective vector $\mathbf{J}_R$ is represented by an executable scoring contract. This contract specifies objective expressions, objective directions, weights, primary progress metrics, and guardrail constraints. Thus, although the main text writes the optimization compactly in minimization form, the executable implementation can represent both minimized and maximized criteria through direction metadata or equivalent loss transformations.

Algorithm 2 Lower-Level Solver: NSGA-II for Fixed Architecture

Require: Architecture d, objective contract $\mathbf{J}_R$, parameter domain $\Theta(d)$, guardrails $\{g_k\}_{k\in\mathcal{K}}$, evaluation budget B, population size N

Initialize parent population $\mathcal{Q}_0 \subset \Theta(d)$ of size N
Evaluate $\mathbf{J}_R(d,\theta)$ and guardrail violations for all $\theta \in \mathcal{Q}_0$
Generate offspring $\mathcal{Q}'_0$ by crossover and mutation
$g \leftarrow 0$
while evaluations $< B$ **do**
  $\mathcal{R}_g \leftarrow \mathcal{Q}_g \cup \mathcal{Q}'_g$
  Evaluate unevaluated offspring by simulation
  Sort $\mathcal{R}_g$ into non-dominated fronts using constraint-dominance
  Fill $\mathcal{Q}_{g+1}$ front by front until the next full front would exceed N
  Complete $\mathcal{Q}_{g+1}$ by crowding-distance selection within the boundary front
  Generate $\mathcal{Q}'_{g+1}$ by crossover and mutation
  $g \leftarrow g+1$
end while
return the non-dominated feasible set $\mathcal{P}^*_d$, or the least-violating frontier if no feasible point is found

The admissible tuning space $\Theta(d)$ is not treated as a free-form set proposed solely by the LLM. Candidate architectures and tuning surfaces are proposed at the upper level, but lower-level optimization is restricted to variables admitted by the structural specification, parameter metadata, package-level declarations, and validation checks. This distinction preserves the methodological role of the LLM as a proposal mechanism while assigning final admissibility to the executable specification.

The framework does not require one lower-level solver for every downstream task. Part 1 and Parts 2.1–2.2 use mixed-variable NSGA-II with feasibility-aware non-dominated selection, while Part 2.3 uses seeded bounded random search under a fixed per-review budget. In both cases, candidate solutions are evaluated through simulation under the declared objective and guardrail contract. The Part 2.3 instantiation illustrates the solver flexibility stated in Section 2.1.

The archive used for upper-level feedback is run-scoped. It stores evaluated non-dominated candidates under the active objective contract and selects representative exemplars for the next proposal round. The archive therefore functions as a validated search memory rather than an unrestricted conversational history.

The distinction between construction and downstream use is important for interpreting the experiments. A calibration experiment directly tests digital-twin construction because the target system and evidence define the twin to be built. A downstream policy or strategy experiment may reuse a calibrated twin as a simulator and may still have a bi-level search structure, but it should be interpreted as a DT-enabled optimization task unless the experiment explicitly constructs a new twin under the same structural-validity and objective-contract assumptions.