Hybrid Methodologies for Observability and AI-Assisted Formal Verification in Concurrent Multi-Agent Systems

Tiberiu-Alexandru Badoi, Andrei Olaru
Faculty of Automatic Control and Computers, *National University of Science and Technology Politehnica Bucharest*
Bucharest, Romania
badoitiberiualexandru@gmail.com, andrei.olaru@upb.ro

***Abstract*—Debugging concurrent Multi-Agent Systems (MAS) is inherently challenging due to non-determinism, asynchronous scheduling, and the tight interleaving of multi-threaded shared states. Traditional debugging methods often fail to capture the semantic intent of agent interactions or introduce significant performance overhead, known as the "Probe Effect." This research proposes a generalized Hybrid Observability Framework for concurrent agent environments. The proposed methodology combines a low-overhead, asynchronous data ingestion layer with a neuro-symbolic analysis engine powered by Large Language Models (LLMs). Empirical results gathered via the FLASH-MAS platform demonstrate that an event-driven, lock-free buffering strategy allows for high-fidelity data capture with negligible impact on simulation timing. Furthermore, we formalize and evaluate an adaptive simulation protocol via a Reactive Hot-Swap by Cascade Propagation mechanism, designed to dynamically isolate non-deterministic cognitive entities while compressing global computational execution costs.**

***Index Terms*—Multi-Agent Systems, Simulation Observability, Runtime Verification, Neuro-Symbolic AI, Adaptive Simulation, Hot-Swapping.**

I. Introduction

Multi-Agent Systems (MAS) have emerged as a dominant architectural paradigm for engineering complex, concurrent software ecosystems. By decomposing computationally intensive problems into highly interconnected networks of autonomous computational threads, MAS frameworks offer significant structural robustness, parallel efficiency, and modular flexibility. The integration of advanced cognitive components, specifically Deep Reinforcement Learning (DRL) policies and localized Large Language Models (LLMs), has shifted modern MAS implementations from deterministic, rule-based execution toward adaptive semantic reasoning.

While these cognitive entities exhibit superior multi-criteria optimization in uncertain environments, their non-deterministic nature introduces severe challenges to traditional software engineering methodologies, particularly regarding system observability, runtime diagnostics, and formal verification. Understanding behavioral divergence and preventing hidden cascading faults within multi-threaded agent pools demands a foundational software layer capable of capturing high-fidelity telemetry without disrupting the discrete temporal scheduling of active concurrent loops.

Developing and debugging concurrent cognitive agents is governed by major operational hurdles: 1) Non-Determinism and the execution Probe Effect; 2) Cognitive Overload caused by massive, unstructured log arrays; 3) The invalidation of linear trace replay in stochastic decision spaces; and 4) The inadequacy of statically compiled guardrails against emergent strategic sub-optimization.

To bridge these gaps, this paper addresses the following **Research Question (RQ)**: *How can a concurrent multi-agent system isolate and formally verify non-deterministic agent behaviors within a pre-recorded historical environment without inducing global trace desynchronization or prohibitive computational simulation overhead?*

To answer this, we propose a comprehensive, platform-agnostic Hybrid Observability Framework for concurrent actor-based setups. The architecture establishes a unified neuro-symbolic diagnostic pipeline, combining high-fidelity, lock-free telemetry ingestion with probabilistic semantic analysis powered by localized LLM execution and deterministic Finite State Machine (FSM) validation. Additionally, to tackle state drift during single-agent resimulation, we introduce a Reactive Hot-Swap mechanism designed to dynamically balance execution high fidelity with execution speed. To demonstrate the practical viability of the methodology, the framework is instantiated and validated within the FLASH-MAS platform, showing substantial performance advantages across distinct operational profiles.

The rest of this paper is organized as follows: Section II discusses the related work in agent observability. Section III details the proposed architectural design and the formalization of the cascade propagation protocol. Section IV presents the empirical evaluation and experimental metrics, while Section V concludes the paper.

II. Related Work

The architectural paradigm of concurrent system diagnostics has undergone rapid structural evolution, shifting from the manual inspection of localized text logs to automated observability pipelines. This section reviews the literature across foundational concurrent theory, runtime verification, and generative AI diagnostics.

A. Foundations of Concurrent System Observability

The foundational challenge in analyzing multi-threaded agent frameworks stems from the high-density interleaving of execution events across independent, parallel thread pools. Reconstruction of mathematically accurate causal histories is an essential prerequisite for deterministic debugging activities within multi-agent environments.

In his seminal theoretical work, Lamport established that independent execution counters require strict ordering criteria in parallel computing environments [1]. To mitigate this, he introduced logical clocks designed to define a partial ordering based on the "happened-before" relation [1]. This paradigm was refined by Mattern through Vector Clocks, providing the necessary mathematical framework to explicitly differentiate between causally dependent transactions and concurrent operations running on symmetric multiprocessing hardware [2]. This structural differentiation is paramount for intercepting non-deterministic race conditions and memory visibility visibility windows within asynchronous agent communication loops.

In software engineering workflows, these theoretical axioms have been encapsulated into application tracing and profiling engines [3]. Currently, enterprise computing architectures leverage the open-source OpenTelemetry standard to aggregate cross-thread traces, performance metrics, and application logs into unified analytical backends [4]. However, while industrial tracing standards provide high visibility into thread allocation and call stack boundaries, they treat the internal cognitive loops of autonomous actors as opaque black boxes, requiring heavy custom instrumentation to expose deep behavioral states.

B. Automated Trace Analysis and Runtime Verification

Once baseline telemetry is ingested, the computing challenge transitions to automated verification. Historical methodologies have divided into rigid formal verification and statistical machine learning log analytics.

Runtime Verification (RV) engines, notably the Monitoring-Oriented Programming (MOP) framework, allow systems to validate active execution traces against definitions specified via Linear Temporal Logic (LTL) formulas [5]. While mathematically rigorous, runtime verification methods are severely bottlenecked by their deterministic nature; they require exhaustive expert knowledge to map out state-transition matrices and are entirely blind to strategic drifts that do not violate raw syntactic boundaries.

To transcend the rigidity of rule-based verification, statistical deep learning paradigms treat logs as semantic token arrays. The DeepLog framework utilized Long Short-Term Memory (LSTM) networks to learn the underlying grammar of normal workflows, flagging statistical deviations as anomalies [6]. Similarly, advanced fault-localization engines construct causal graphs across concurrent components by applying causality inference to telemetry metrics [7]. Despite their pattern-recognition performance, deep learning log parsers lack structural semantic understanding; they process text streams as abstracted vectors, rendering them incapable of interpreting high-level negotiation failures.

C. Generative AI and Agentic Diagnosis

The recent paradigm shift in generative artificial intelligence has catalyzed the transition from statistical pattern matching to cognitive semantic reasoning. Software engineering validation studies indicate that Large Language Models (LLMs) significantly outperform highly specialized log parsers in structural extraction tasks. Architectures like LogGPT demonstrate that pre-trained foundation models can execute zero-shot log parsing with extreme accuracy, mapping variable parameters inside complex text templates without requiring domain-specific training regimens [8].

However, a prohibitive barrier preventing the direct application of LLM cores within large-scale multi-agent simulation analysis is the absolute restriction of the model's prompt context window. Concurrent simulations generate dense text telemetry, triggering attention degradation in generative architectures. To circumvent this, recent frameworks introduce specialized Retrieval-Augmented Generation (RAG) pipelines for incident investigation, vectorizing execution logs and dynamically retrieving relevant snippets [9].

The cutting edge of current computing research focuses on autonomous Agentic Workflows. Advanced software engineering tools, such as SWE-agent, demonstrate that language models can function as active diagnostic agents when equipped with specialized technical toolkits, including custom file system readers and automated runtime access [10]. In this context, the AI system does not merely summarize static log lines; it actively interacts with the application context, dynamically formulating diagnostic hypotheses and validating them directly against the primary codebase [10], [11].

D. Identification of the Scientific Research Gap

Despite technological acceleration across concurrency tracing, formal verification, and agentic AI, a profound systemic gap persists in the literature. There is a lack of unified, non-invasive frameworks capable of bridging the absolute mathematical certainty of deterministic recording with the advanced cognitive reasoning of Agentic AI specifically engineered for object-oriented concurrent Multi-Agent Systems. Existing tools force developers to choose between brittle manual LTL modeling or probabilistic, hallucination-prone output of standalone generative models. Our framework bridges this gap by establishing an integrated neuro-symbolic diagnostic pipeline. A structural comparison of our framework against the state of the art is formalized in Table I.

TABLE I
STRUCTURAL COMPARISON OF VERIFICATION FRAMEWORKS

Methodology	Probe Effect Resistance	Semantic Reasoning	Formal Guarantees	Environment
Vector Clocks [2]	High (Theoretical)	None	Absolute	Concurrent
OpenTelemetry [4]	Medium (Sync I/O)	None	None	Multi-threaded
Java MOP [5]	Low (Instrumentation)	None	Absolute	Java Threads
DeepLog [6]	Medium (Async Thread)	Statistical	Probabilistic	Static Logs
RCACopilot [9]	N/A (Offline Analysis)	High	None	Concurrent Arch
Proposed	**High (Lock-Free)**	**High (LLM)**	**Absolute (FSM)**	**Platform Agnostic**

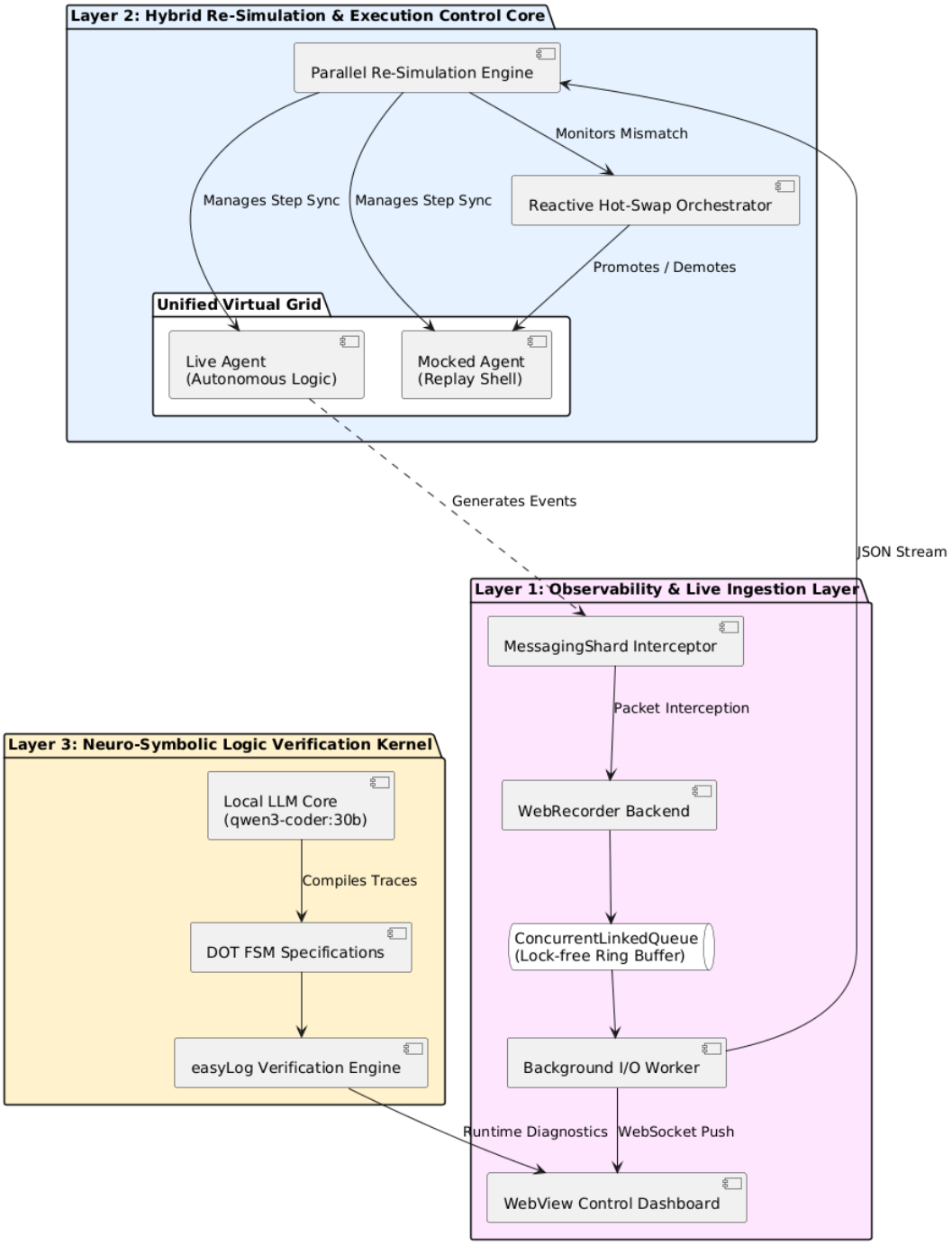


Fig. 1. The three-tier architecture of the Hybrid Observability and Neuro-Symbolic Verification Framework.

III. Problem Statement and Proposed Approach

To isolate and analyze a modified cognitive agent (the *Live Subject*) without executing hundreds of other heavy AI entities concurrently, our middleware isolates the subject inside a pre-recorded environment composed of *Mocked Agents* that replay event streams from a historical dataset [15]. However, this induces the *Cognitive Drift* paradox: once the live subject makes a decision that deviates from the baseline (the **Behavioral Divergence Point**), the surrounding mocked agents remain blind to the new payload, desynchronizing the global simulation context.

A. Proposed Solution: Reactive Hot-Swap by Cascade Propagation

To resolve the Cognitive Drift paradox without reverting to resource-heavy full-live runs, we formulate an adaptive runtime synchronization pattern: the *Reactive Hot-Swap by Cascade Propagation* algorithm. The goal of this model is

to dynamically construct a *Transient Live Execution Bubble* around the localized zone of behavioral divergence.

Let the concurrent agent interaction configuration be represented as a directed graph $G = (A, M)$, where $A = \{a_1, a_2, ..., a_n\}$ is the finite set of active agent threads, and $M \subset A \times A$ represents communication channels. We define the execution state of any agent a_i at a discrete logical step t as:

$$\sigma_i(t) \in \{\text{Mocked}, \text{Live}\} \tag{1}$$

At $t = 0$, the system initializes such that $\sigma_{\text{target}}(0) = \text{Live}$ and $\sigma_i(0) = \text{Mocked}, \forall a_i \neq a_{\text{target}}$. Let $E_{\text{hist}}(a_i, t)$ be the historical payload recorded in the log database, and $E_{\text{actual}}(a_i, t)$ be the actual payload computed during the hybrid run. A Behavioral Divergence Event (Δ) is triggered at step t if and only if:

$$\exists a_i \in A \text{ s.t. } \sigma_i(t) = \text{Live} \wedge E_{\text{actual}}(a_i, t) \neq E_{\text{hist}}(a_i, t) \tag{2}$$

When Δ is flagged, the cascade orchestration design executes the following recursive runtime operations:

1) **Undeclared State Capture:** The central observation layer flags the divergence and updates the structural state to `DIVERG`.
2) **Neighborhood Extraction:** The engine queries the topology graph to extract immediate transactional neighbors:
$$\mathcal{N}(a_i) = \{a_j \in A \mid (a_i, a_j) \in M \vee (a_j, a_i) \in M\} \tag{3}$$
3) **Hot-Swap Execution Strategy:** For each neighbor $a_j \in \mathcal{N}(a_i)$, if $\sigma_j(t) == \text{Mocked}$, its core execution state is initialized, memory is synchronized via a `SharedStateBridge`, and its mode is promoted to $\sigma_j(t) \leftarrow \text{Live}$.
4) **Recursive Wave Propagation:** Step 3 propagates recursively along the interaction dependencies if newly activated agents alter downstream choices as a reaction to the original divergence.

B. The Convergence and Re-Alignment Phase

At each subsequent step $t + k$, the framework evaluates the active bubble entities. Re-alignment occurs when the autonomous sub-network naturally stabilizes and matches the historical trajectories again:

$$\forall a_j \in \text{Bubble}, \quad E_{\text{actual}}(a_j, t + k) == E_{\text{hist}}(a_j, t + k) \tag{4}$$

The moment this mathematical equality is fulfilled, the engine executes an automated **Mass Demotion** ($\sigma_j(t + k) \leftarrow$ Mocked), collapsing the active execution bubble back to a low-overhead resource replay state.

C. Ecosystem Architecture Layers

The framework manages a modular three-tier architecture designed to detach logging mechanics from underlying platform components:

- **Layer 1: Observability and Live Ingestion Layer:** Captures message streams directly from the memory shard. Thread-segregated operations utilize lock-free ring buffers to keep core thread release latencies below 100ns, reverting to a short-circuited compiled pass when logging is disabled.
- **Layer 2: Hybrid Re-Simulation Core:** Manages step execution and hosts the runtime manager responsible for triggering cascade hot-swaps and deploying state hydration bridges.
- **Layer 3: Neuro-Symbolic Logic Verification Kernel:** Ingests raw telemetry streams, utilizing localized LLMs (`qwen3-coder:30b`) to compile single-agent traces into Graphviz `.dot` formal state machines. These are validated by a regex-based parser engine, removing AI hallucinations from the formal verification loop.

IV. Empirical Evaluation and Discussion

To evaluate the operational footprint, scalability, and structural flexibility of the proposed generalized architecture, the framework was instantiated as a middleware prototype and evaluated within a concrete multi-agent environment. The FLASH-MAS engine [13], [14] was selected as the experimental validation platform due to specific architectural properties that align with our methodology: its lightweight Java thread-scheduling model, isolated memory shards, and entirely asynchronous event-driven messaging layer. These characteristics provide an ideal, non-invasive environment for intercepting message components and deploying state hydration bridges, demonstrating that the underlying hybrid verification theory can be mapped onto modern concurrent software runtime architectures.

The experimental case study used for this evaluation models a highly concurrent *Transactive Energy Market* micro-grid system [15]. The validation environment consists of 3 localized market coordinator threads (`MarketAgent`) and 99 concurrent micro-generation asset components (`ProsumerAgent`) executing stochastic bidding optimization policies [16], [17]. The workload profile requires the multi-threaded system to execute a continuous global threshold of 10,000 completed market transactions.

A. Observability Pipeline Performance

The first experiment isolates the programmatic friction introduced by the logging pipeline. Test 1 evaluates the system with the logging pipeline completely disabled via a short-circuited compiled pass, while Test 2 deploys the active asynchronous recording engine writing JSON records to persistent storage. Both tests run in a 100% `Live` environment under unbounded CPU execution loop parameters. The empirical results are formalized in Table II.

TABLE II
Operational Telemetry Overhead Metrics

Configuration Profile (10,000 Trades)	Duration (ms)	Avg JVM CPU Load
Test 1: Logging DISABLED (All Live)	11,771	6.40%
Test 2: Logging ENABLED (All Live)	13,997	9.34%

The empirical profiles demonstrate that the active logging pipeline introduces a contained temporal penalty of 18.9% ($\Delta = 2,226$ ms), alongside an expected expansion in average processor allocation from 6.40% to 9.34%. This low footprint validates the lock-free ring buffer design; by decoupling text serialization and I/O write loops from the main concurrent scheduling frames, the framework successfully minimizes execution Probe Effect distortions during dense processing bursts.

B. Scalability and Dynamic Hot-Swapping Metrics

The second evaluation phase critically assesses the computational efficiency of the adaptive resimulation protocol by measuring performance limits under structural workload partitions. We configure and compare three independent network topologies over the identical 10,000 transaction threshold under unbounded processing parameters:

- **Test 3 (Partial Emulation Profile):** Microgrid Zone A executes completely in a `Live` environment (34 active structural nodes), while the remaining 66 decentralized prosumer threads across Zones B and C are bound to passive, reactive `MockPlaybackShell` instances driven by historical inputs.
- **Test 4 (Dynamic Boundary Mutation Sequence):** The environment initializes under identical boundary constraints as Test 3, but a programmatic behavioral divergence is forced at simulation step 15, triggering an immediate promotional hot-swap that mutates the 66 placeholder shells into active cognitive reasoning loops.
- **Test 5 (Full Structural Emulation Baseline):** The macro-grid environment is completely locked inside passive historical replay configurations, bypassing cognitive model evaluation across all 99 prosumer nodes to define the absolute operational baseline floor.

The empirical scaling metrics are formalized in Table III.

TABLE III
SCALABILITY PROFILES UNDER WORKLOAD PARTITIONS

Simulation Workload Profile (10,000 Trades)	Duration (ms)	Avg JVM CPU Load
Test 3: Partial Emulation Profile (66/99 Mocked)	6,178	7.63%
Test 4: Dynamic Boundary Mutation Sequence	11,571	6.43%
Test 5: Full Structural Emulation Baseline	1,537	17.21%

The scaling results confirm a highly linear behavior across execution topologies. When 100% of the cognitive agent loops are bypassed (Test 5), the system processes the entire 10,000 transaction threshold in just 1,537 ms, saturating the CPU at 17.21% due to dense, non-blocking asynchronous memory scheduling. Shifting 1/3 of the infrastructure into active behavioral models (Test 3) expands the duration to 6,178 ms, demonstrating a near-perfect linear scaling pattern ($\approx 50\%$ savings in total execution time compared to full live runs).

Crucially, the dynamic promotion pipeline (Test 4) successfully transitions state matrices mid-execution without introducing computational locks or core messaging stalls. The execution horizon stabilizes at 11,571 ms, confirming that the runtime overhead of promotional thread mutation is negligible. Keeping decentralized sectors inside a transient mocked bubble yields massive performance gains, safely offloading execution costs until live promotion is strictly demanded by behavioral anomalies.

V. CONCLUSION AND FUTURE WORK

This research detailed a platform-agnostic Hybrid Observability and Re-simulation Framework engineered for concurrent multi-agent systems. By integrating an asynchronous lock-free ingestion abstraction with a neuro-symbolic verification layer, the architecture successfully minimizes logging overhead while extracting formal behavioral states. The integration of a Reactive Hot-Swap by Cascade Propagation mechanism resolves the paradox of Cognitive Drift by dynamically expanding active execution bubbles only when behavioral divergence occurs. Empirical validation on the FLASH-MAS platform confirms that hybrid agent environments achieve high execution elasticity, cutting simulation overhead by half under partial mock configurations. Future research will focus on transitioning log analysis to high-dimensional semantic embeddings to enable similarity clustering and establishing automated root-cause agent diagnostics.

REFERENCES

[1] L. Lamport, "Time, clocks, and the order of events in a distributed system," *Commun. ACM*, vol. 21, no. 7, pp. 558–565, 1978.

[2] F. Mattern, "Virtual time and global states of distributed systems," *Parallel Distr. Algor.*, vol. 1, no. 23, pp. 215–226, 1988.

[3] B. H. Sigelman et al., "Dapper, a large-scale distributed systems tracing infrastructure," Google, Tech. Rep. 2010-1, 2010.

[4] The OpenTelemetry Authors, "OpenTelemetry: High-quality, ubiquitous, and portable telemetry," Cloud Native Computing Foundation, 2024. [Online]. Available: https://opentelemetry.io/

[5] F. Chen and G. Rosu, "MOP: an efficient and generic runtime verification framework," in *Proc. OOPSLA '07*, 2007, pp. 569–588.

[6] M. Du, F. Li, G. Zheng, and V. Srikumar, "DeepLog: Anomaly Detection and Diagnosis from System Logs through Deep Learning," in *Proc. CCS '17*, 2017, pp. 1285–1298.

[7] Y. Meng et al., "MicroCause: Localizing Failure Root Causes in a Microservice through Causality Inference," in *Proc. IWQoS*, 2020, pp. 1–10.

[8] V. H. Le and H. Zhang, "Log Parsing with Large Language Models," in *Proc. ESEC/FSE '23*, 2023.

[9] H. Chen, Y. Liu, X. Zhang, and J. Yang, "RCACopilot: Automated Root Cause Analysis Assistant for Cloud Incidents with LLMs," in *Proc. ASE '24*, 2024.

[10] J. Yang et al., "SWE-agent: Agent-Computer Interfaces for Automated Software Engineering," *arXiv preprint arXiv:2405.15793*, 2024.

[11] Z. Liu et al., "LILAC: Log Parsing using LLMs with Adaptive Context," in *Proc. SANER '24*, 2024.

[12] T. C. Ciot, "easyLog: A General-Purpose Verification Framework for Complex Systems," Dissertation Thesis, Politehnica Bucharest, 2025.

[13] A. Olaru, A. M. Florea, and F. Radu, "Making FLASH-MAS: a modular and lightweight agent platform," in *Proc. International Workshop on Agent-Based Technologies and Applications*, 2015.

[14] A. Olaru, "Ambient Intelligence with FLASH-MAS," in *Proc. European Conference on Multi-Agent Systems (EUMAS '25)*, Springer, 2025.

[15] E. Mengelkamp et al., "Designing microgrid energy markets: A case study: The Brooklyn Microgrid," *Appl. Energ.*, vol. 210, pp. 870–880, 2018.

[16] M. R. Alam, M. St-Hilaire, and T. Kunz, "Peer-to-peer energy trading among smart homes," *Appl. Energ.*, vol. 238, pp. 1434–1443, 2019.

[17] J. P. Krogh and S. B. Agner, "Multi-agent reinforcement learning for transactive energy systems: A comprehensive review," *IEEE Access*, vol. 12, pp. 4520–4545, 2024.