A 3-tiered agentic AI Framework for Verification Regression

Jin Choi[1], Sangwoo Noh[2], Seonghee Yim[3], Youngsik Kim[4]
([1]jjin.choi; [2]sangwoo.noh; [3]sh.yim; [4]ys31.kim@samsung.com)

***Abstract* - As modern System-on-Chip (SoC) designs continue to grow in complexity, verification regression faces increasing challenges in scalability, continuous integration (CI) management, and redundant debugging efforts. Conventional monolithic regression systems repeatedly execute massive test sets without considering hierarchical design boundaries or verification scope, resulting in high latency, limited scalability, and inefficient resource utilization. To address these limitations, this paper proposes a 3-tiered agentic AI regression framework that structures verification regression into three hierarchical tiers—user, block, and central regression tiers—each aligned with a distinct verification scope and purpose. The user regression tier formalizes engineer-initiated sanity regressions as a lightweight yet explicit verification tier. Meanwhile, block regression tier performs change-aware regressions triggered by design and verification updates to detect block-level corner cases in the early stage. The central regression tier conducts chip-level regressions with automated failure classification and ownership routing. To coordinate these tiers, the proposed framework leverages modular agent-based components including regression type agents, bridge type agents, and operation type agents, which collectively e regression execution and cross-tier communication. This agentic design allows regression behavior to be dynamically adapted according to project context, team-specific verification scope and evolving verification objectives, rather than being constrained by static pipelines. By organizing distributed regression activities into a hierarchical structure, the proposed framework provides a consolidated and data-driven view of verification progress across teams and environments. Moreover, enforcing verification coverage across all three tiers reduces the risk of overlooking critical verification holes, leading to a more robust and systematic verification process. Instead of solely optimizing turnaround time, the framework emphasizes improved regression governance, traceability, and informed decision-making throughout the verification stages. This paper establishes a practical foundation for scalable, adaptive, and self-managing regression infrastructures suitable for large-scale SoC verification.**

I. INTRODUCTION

Modern System-on-Chip (SoC) designs have reached unprecedented levels of complexity, integrating tens of millions of gates and hundreds of interdependent IP blocks. Consequently, regression has become increasingly critical in design verification, as it ensures functional correctness and stability across continuous design iterations by repeatedly executing large numbers of test cases. Given this repetitive execution at scale, verification regression is inherently resource-intensive, motivating significant ongoing efforts to improve its efficiency and automation. A wide range of approaches have been proposed to streamline regression execution and management. In industrial practice, Jenkins-based continuous integration (CI) approach is widely used to schedule daily or weekly regressions or to automatically trigger regression runs upon code changes, thereby reducing manual execution overhead and improving regression availability [1]. Other studies focus on optimizing specific aspects of the regression workflow, such as intelligent failure triage and ownership assignment using machine learning techniques [2]. More recently, large language models (LLMs) have been explored to summarize regression logs, extract failure patterns, and enhance debugging productivity through retrieval-augmented generation (RAG)–based data access [3]. While these approaches provide meaningful improvements, they largely operate within the constraints of existing monolithic regression infrastructures. Most existing solutions target localized optimization of individual regression stages—such as scheduling, failure triage, or log analysis—without addressing the underlying structural limitations of regression itself. As a result, regression systems remain flat and centralized, executing broad test cases without adapting verification scope to design hierarchy, change locality, or team-specific responsibilities.

These limitations become more pronounced in large-scale verification environments where verification activities are distributed across multiple teams and heterogeneous infrastructures. Under these conditions, maintaining a unified and consistent view of verification regression status is challenging, and aggregating results across environments often requires significant manual intervention. Furthermore, although emerging technologies such as LLMs, agent-based AI, and Model Context Protocol (MCP) servers offer strong potential to improve verification workflows, embedding these technologies into a monolithic regression pipeline fundamentally limits scalability, extensibility, and long-term maintainability. As verification environments continue to grow in scale and complexity, this structural mismatch between monolithic regression architecture and emerging verification workflows fundamentally limits efficiency and introduces additional verification overhead.

To address these challenges, this paper proposes a 3-tiered agentic AI regression framework that restructures verification regression from a system-level perspective. The proposed framework introduces a three-level hierarchical regression structure comprising user, block, and central tiers, each defined as a logical abstraction of verification granularity with a specific scopes and objectives in this paper. This 3-tiered verification regression is performed progressively, with results dynamically promoted and demoted across tiers rather than being executed through a single monolithic regression pipeline. The architecture is designed around a publish-subscribe-based tiered architecture combined with modular AI agents, which enables loosely coupled coordination across tiers via asynchronous pub-sub mechanisms. In the proposed framework, agentic AI components are embedded to orchestrate regression tasks, manage inter-tier communication, and adapt verification behavior to project context and team-specific requirements. By distributing regression intelligence across specialized agents rather than concentrating it in a centralized pipeline, the proposed framework provides a scalable and maintainable architecture that improves regression governance and reduce the risk of verification gaps across distributed environments. In addition, this architecture establishes a robust foundation for integrating advanced AI technologies into verification workflows.

II. PROPOSED METHODS

This paper proposes a hierarchical regression framework that structures verification workflow through explicitly defined tiers and communication interfaces. The framework enables each tier to fulfill a specific verification role and collectively forming a robust and systematic verification process by separating regression activities into distinct tiers within a scalable system architecture. The proposed framework further classifies agents into three functional categories according to their roles and communication models: regression, bridge and operation type agents. Regression type agents perform and manage regression tasks within each tier, while bridge type agents act as a mediator between regression tiers through a publish-subscribe mechanism, enabling hierarchical coordination. Action type agents execute auxiliary functions such as test dispatching and monitoring. This classification provides both modularity and scalability: Regression agents concentrate on tier-specific verification goals, bridge agents synchronize control and data exchange across tiers, and operation agents handle operational tasks in distributed environments.

A. Agent typology

Regression type agents represent the core verification intelligence within each regression tier. Their primary responsibility is to determine which test cases should be executed, how regression outcomes are interpreted, and when test cases should be promoted or deferred across tiers. Although their scope and execution context differ by tier, all regression type agents share the common role of managing tier-specific verification objectives. At the user level, we propose the user regression agent executes engineer-initiated sanity regressions to provide feedback in the timely manner. The block regression agent operates at the block level, performing targeted and randomized regressions in response to design changes and verification maturity to assess block-level stability. At the chip-level, the central regression agent conducts full-chip integration regressions and performs structured error classification to support failure triage. In the proposed framework, these regression agents are designed not to communicate directly with one another. Instead, each agent registers with a corresponding bridge type agent, which serves as a publish-subscribe hub to mediate information exchange and coordination between regression tiers. This design choice preserves tier isolation while enabling controlled and hierarchical propagation of regression results.

We introduce ***bridge type agents*** as dedicated coordination components that mediate communication across regression tiers. By routing inter-tier interactions through bridge agents, the framework decouples tier-specific logic while enabling hierarchical coordination within the overall regression system. Regression type agents register with their corresponding bridge agents and exchange regression results and updates through an asynchronous publish-subscribe mechanism. This design allows regression activities at different tiers to progress independently without tight structural coupling. The framework employs two bridge agents: The user-block bridge agent governs interactions between user and block regression tier by forwarding validated results upward while redirecting failing test cases downward for localized re-validation. The gate agent, positioned between block and central regression tier, applies promotion criteria by allowing only blocks that satisfy predefined stability thresholds ensuring that unresolved issues remain confined to their appropriate verification scope. Through this bridge-based structure, the proposed framework separates regression execution from verification responsibilities and enables regression instances to be attached or detached dynamically supporting concurrent execution across multiple tier regressions.

This paper introduces ***operation type agents*** to handle operational and integration tasks that support regression workflow across all tiers. Unlike regression type agents which are responsible for verification decision-making, operation type agents are designed to execute concrete system-level functions such as test case dispatch, error classification, and interaction with external infrastructure. These agents are intentionally kept stateless and are invoked on demand, allowing them to operate independently of tier-specific verification logic. The proposed architecture

separates verification control logic from execution and integration concerns by structuring operational responsibilities. This separation improves modularity, simplifies extension of operational capabilities, and allows regression workflows to scale without impacting the core regression system.

B. 3-tiered regression architecture

The proposed framework is organized into three tiers: user, block, and central tier. These tiers reflect the natural expansion of verification scope observed in practice, starting from engineer-local validation and gradually progressing toward full-chip integration. Each tier operates independently but participates in a controlled escalation process governed by inter-tier communication. Communication and synchronization across tiers are mediated by bridge, which implement a publish-subscribe model to enable asynchronous coordination while preserving functional independence between layers.

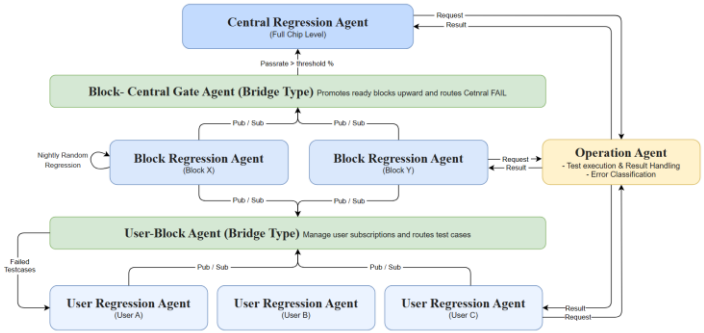

Figure 1. A 3-tiered Regression Framework Architecture

Figure 1 illustrates the concurrent operation of multiple user regression agents (e.g., user A, user B) that each representing an individual engineer's verification environment. Each user regression agent subscribes to the user–block bridge agent, which serves as a coordination layer that aggregates execution results and publishes validated test cases to one or more block regression agents (e.g., block X, block Y). The block regression tier manages block-specific regressions using design-aware and randomized strategies continuously assessing block stability. The gate agent promotes the verified blocks to the central regression agent for full-chip regression once predefined stability criteria are satisfied.

The ***user regression tier*** represents individual engineer environments as the first line of verification, where multiple user regression agents operate concurrently, each aligned with a specific user or development workspace. In practice, engineers execute sanity regressions repeatedly to validate incremental design modifications. In many cases, identical or similar test sets are re-executed within short time intervals. As a consequence, user-initiated regressions exhibit execution patterns—such as frequency, pass rates, and completion latency —that differ fundamentally from those of block- or chip-level regressions. To accommodate these characteristics, the user regression tier is deigned to focus on enabling systematic handling of repetitive sanity regressions while minimizing overhead for verification engineers. Accordingly, this tier is optimized for fast turnaround and frequent re-execution, prioritizing low-latency functional feedback over exhaustive coverage. This tier is realized through user regression agents which derive quantitative summaries of regression in a measurable form by tracking sanity regression frequency, failure signatures, and average execution latency. Thereby, these agents provide a structured interface between engineer-driven regression activity and subsequent regression tiers. By modeling engineer-initiated sanity regression as an explicit tier, the proposed framework converts an informal and tool-dependent practice into a controlled and analyzable component of the overall regression system. This separation improves transparency and productivity across regression and provides a stable basis for hierarchical promotion to subsequent verification.

During the early stage of verification, design and verification related updates occur frequently. Yet, verification engineers often lack sufficient capacity to manually identify affected test cases and execute them at each release point. As a result, regressions introduced by incremental changes may persist unnoticed across multiple releases, making it difficult to determine when and where failures were introduced. The accumulation of such unresolved issues significantly increases the complexity of late-stage debugging at the chip level. To address this challenge, the proposed architecture introduces the ***block regression tier*** as an intermediate verification stage between user-driven sanity regression and chip-level regression. The primary objective of this tier is to validate block-level changes based on detected updates and assess stability before advancing to chip-level verification. By constraining regression to the block scope, this tier enables more thorough validation of change-affected logic while avoiding unnecessary chip-wide execution.

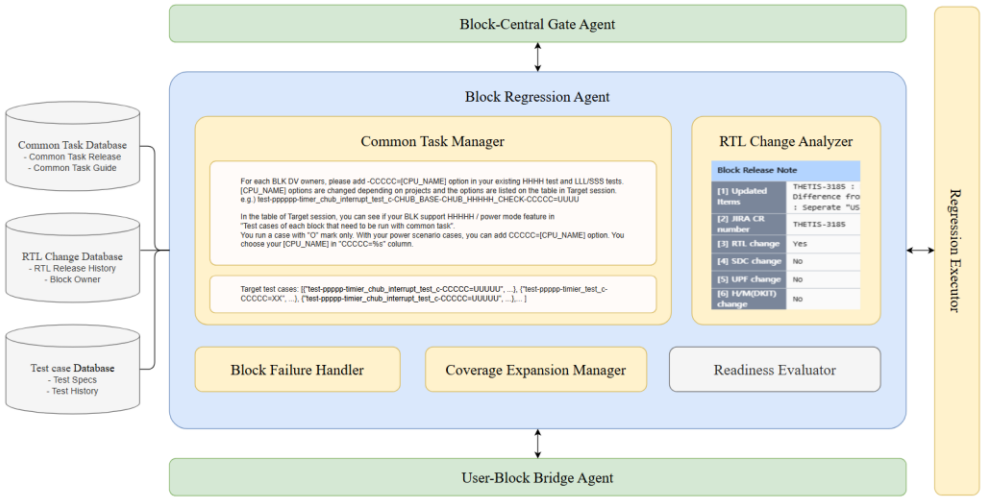

Figure 2. Block Regression Tier

The block-level regression is operated through the block regression agent, which serves as the execution and control mechanism of this tier as illustrated in Figure 2. The agent continuously monitors RTL and common task release information, where common tasks define shared verification requirements applied across multiple blocks, and analyzes release notes to identify affected design blocks. Based on the documented changes, the agent derives the corresponding block-scoped test cases and required execution options, ensuring that all change-affected scenarios are executed. When necessary, randomized execution is incorporated to expand block-level coverage and reduce bias toward a limited set of previously exercised scenarios. When failures are detected, the agent confines failure analysis to the modified block and reports execution results with contextual information to the corresponding block owner. In addition, the agent evaluates block-level execution completeness and stability to determine qualification criteria for central regression, preventing unstable or incomplete results from propagating to higher regression tiers.

The proposed framework clearly separates verification responsibility by introducing block-level regression as a dedicated verification and this tier automates repetitive and error-prone manual tasks such as executing test cases impacted by detecting and interpreting modifications. This enforces execution completeness and qualification constraints at the block level, ensuring that block-level validation is performed in a consistent and well-defined manner. As a result, the framework improves the quality and clarity of subsequent chip-level regression, reducing noise caused by unresolved block-level issues and enabling more effective system-level failure analysis.

The ***central regression tier*** is responsible for chip-level verification with a focus on multi-block interactions and chip-level integration. In the proposed framework, execution at this tier is designed to be initiated after block-level pass rates reach a user-defined threshold, typically between 90% and 95%, reflecting sufficient block-level stability. This gating strategy ensures that central regression resources are allocated to avoid premature chip-level execution and reduce unnecessary resource consumption caused by unresolved block-specific issues. At this tier, regression execution and failure handling are coordinated by a set of specialized agents that manage the complexity of full-chip verification. An error classification agent analyzes regression failures and distinguishes between common task errors and domain-specific errors, enabling structured and consistent failure categorization. Based on this classification, the ownership routing agent maps each failure to the appropriate owner and propagates summarized diagnostic information including failure signatures and contextual metadata by issue registration in an external issue tracking system (e.g., JIRA). The agents in central regression tier reduce manual intervention while preserving end-to-end traceability throughout the verification regression workflow.

C. Publish-subscribe interaction

The proposed framework adopts a message queue–based publish–subscribe model implemented using RabbitMQ to coordinate regression activities across the user, block, and central tiers. This design aligns with the hierarchical and asynchronous characteristics of large-scale verification regression in which each tier operates at a different verification scope and objective. The user regression agents continuously publish sanity regression outcomes without being synchronized with block-level execution while the block regression agents consume validated results only when resources and stability conditions permit. The central regression is further isolated from transient activity at lower tiers and is triggered solely through explicit promotion events issued by the gate agent. This separation allows each tier to progress independently while remaining logically connected.

The architecture also functions as a buffering and isolation boundary between tiers, preventing sudden bursts of user-initiated regressions from overloading block-level infrastructure and ensuring that block re-executions triggered by RTL design changes do not interfere with ongoing user workflows. The proposed architecture use topic exchanges and routing keys to selectively deliver events to bridge agents while acknowledgment-based message handling enables controlled retries and failure isolation. In this framework, the message broker provides the underlying delivery mechanisms, including acknowledgment and redelivery support, while bridge agents apply retry and failure isolation policies derived from verification engineer–defined configurations and runtime regression context to control how regression events are interpreted and propagated across tiers. This mechanism absorbs transient execution or infrastructure issues without propagating instability across tiers, allowing regression progress to remain robust under partial failures or delayed processing. Through this message broker-mediated publish–subscribe architecture, the proposed framework enables asynchronous and controlled verification effort across hierarchical regression tiers, providing a scalable foundation for managing regression complexity in distributed verification environments.

D. Integration of operation agents

The proposed framework implements operation type agents as execution-oriented components responsible for performing concrete actions. These agents are designed with explicit input and output interfaces and operates in a largely stateless manner. Each operation agent retrieves required execution context from shared databases, while execution results and metadata are persisted upon completion. This design allows operation agents to remain independent of execution history while supporting reliable recovery and reuse across regression cycles.

Within this framework, the regression execution agent materializers regression decisions into executable workloads. Upon receiving an execution request, the agent constructs a job specification that encodes the target block or system scope, selected testcases, execution options, tool versions, and seed policies. This specification is translated into environment-specific execution commands through a dedicated adapter layer, allowing the same agent logic to support heterogeneous verification environments. Execution artifacts are exchanged through a filesystem MCP interface that provides a controlled and standardized file access boundary between the execution environment and the agent. The agent monitors job progress by observing the creation and update of execution outputs within designated filesystem paths. Upon completion, the agent collects execution artifacts including results summaries, simulation logs and coverage data and persists the extracted results into the regression database with a standardized schema via in-house REST API MCP server.

The ownership routing agent consumes classified failure records and determines responsibility based on predefined block-level information database and historical issue data retrieved leveraging an external issue-tracking tool MCP server. Failure records are compared against previously reported issues using failure signatures and similarity matching to identify related past occurrences. In the proposed framework, ownership resolution follows a simple prioritized policy. The agent first consults the block owner database to establish a default responsibility assignment. When a failure exhibits a similarity score exceeding 90% with a previously reported issue, the agent overrides the default mapping and assigns ownership to the most recent matching issue owner. This policy reflects the assumption that highly similar failures are likely to share root causes and can be most efficiently handled by engineers familiar with recent related issues.

These operation agents provide a concrete execution backbone for the proposed framework by performing narrowly scoped and well-defined functions through explicit interfaces to shared databases and external systems. By separating operation-specific agents from regression logic, the proposed framework maintains a clear distinction between regression and execution mechanics. In this paper, the framework incorporates a minimal set of operation agents to validate core functionalities while the modular design permits individual agents to be extended or replaced as verification requirements evolve, enabling incremental enhancement without disrupting the overall regression architecture.

III. Experimental Results

This section presents an empirical evaluation of the proposed 3-tiered agentic AI regression framework, conducted within an industrial SoC verification environment using production-grade design verification datasets to assess its effectiveness in practical verification workflows. The objective of this evaluation is to demonstrate that the hierarchical regression architecture augmented with agentic AI improves scalability and efficiency when compared to traditional monolithic regression pipelines. The evaluation focus on three key aspects: (1) execution completeness and qualification enforcement at the block level, (2) reduction of regression noise and resource waste at the chip level, and (3) the architectural significance and functional contribution of agent-based coordination as demonstrated through the ablation studies.

A. Limitation of traditional chip-level regression

In a traditional regression flow without an explicit block-level regression tier, regression is typically performed either by executing chip-level regressions following RTL design updates or through engineer-driven execution of a subset of test cases selected based on individual judgement. To quantify this limitation of this approach, regression execution status across 33 design blocks was examined before the first chip-level regression was initiated. Out of 20,766 block-level test cases, 2,892 test cases had no recorded execution results at the time chip-level regression was initiated, resulting in an overall unexecuted rate of 13.93%. These unexecuted test cases were unevenly distributed across blocks, with several blocks exhibiting missing execution rates exceeding 25% and one block alone accounting for more than 1,100 test cases that had never been executed. Further analysis showed that these test cases were not intentionally excluded due to low priority; rather it occurred when test cases were not registered due to manual oversight or when required execution options were inadvertently omitted. In addition, results produced from engineer-driven executions are often fragmented across individual environments, making systematic aggregation difficult and introducing inconsistencies in execution conditions and configuration assumptions. As a result, critical failures are often detected only at later stages of regression where overall debugging effort increases and failure localization becomes more difficult.

B. Impact of hierarchical regression

Due to the computational cost of executing full regression across all design blocks, the evaluation was conducted on representative blocks rather than the complete design. The selected blocks exhibit diverse sizes allowing the impact of hierarchical regression to be examined under realistic verification constraints. At the user regression tier, engineer-initiated sanity regressions were continuously observed and recorded. The user regression agent tracked execution frequency, results, and version-related histories which enabling traceability of when test cases transitioned between pass and fail states. At the block regression tier, RTL design changes were detected and immediately triggered block-scoped regressions. In the traditional workflow, verification engineers typically require two to three days to identify affected test cases and re-execute regressions after design changes. In contrast, the block regression agent eliminated this delay by initiating regression execution as soon as release information became available.

The impact of gating conditions became evident at the central regression tier. The central regression was configured with a relaxed promotion threshold of 80% block-level pass rate, allowing system-level execution only after individual blocks reached a minimum level of stability. For one of the blocks, the central regression in the proposed framework was initiated after the block achieved 1,343 passing and 130 failing testcases out of 1,473 total executions, corresponding to a pass rate of approximately 91.2%. For the other block, the central regression began when the block reached 501 passing and 102 failing testcases, yielding a pass rate of 83.1%. In both cases, central regression was launched only after block-level stability had been established. By contrast, the traditional regression flow without hierarchical gating was initiated while blocks remained largely unstable based on the engineer's subjective assessment. In one observed instance, the first block entered regression with 1,206 failing and 238 passing testcases, while the second block entered with 358 failing and 245 passing testcases. This corresponds to an effective pass rate of approximately 23.6%, indicating that chip-level regression was executed on a poorly validated state. Even under optimistic assumptions, nearly half of the testcases entering regression were already failing, resulting in substantial resource waste and limited diagnostic value. These observations highlight the practical impact of hierarchical regression. By containing instability within the user and block tiers, the proposed framework prevents premature chip-level execution and ensures that central regression operates on a substantially cleaner and more mature verification state. This shift reduces unnecessary simulation cost while enabling more effective chip-level failure analysis. The results demonstrate that hierarchical regression is not merely a structural refinement, but a mechanism that directly improves regression efficiency and outcome quality under realistic verification constraints.

C. Ablation study: architectural significance and functional contributions

We conducted a set of controlled ablation scenarios where each selectively disabling one representative agent type to verify the individual impact of each component. Each ablation scenario corresponds to the removal of a specific agent type, allowing us to isolate its functional contribution to the overall regression framework.

No.	Configuration	Description
A1	No block regression agent	Remove block-level autonomous regression.
A2	No user-block bridge agent	Disable pub-sub coordination; regression agents communicate directly.
A3	No execution agent	Replace distributed execution/dependency modules with static scripts.

Table 1. Ablation Study

In the A1 ablation scenario where the block regression agent was removed, autonomous block-level regressions triggered by RTL design changes where completely eliminated including randomized and design-aware executions. Although user-initiated sanity regression results continued to be collected and block-level regression could still be executed manually when explicitly requested by engineers, block-level pass rate accumulation relied primarily on user-initiated regressions. Consequently, block stability evolved more slowly and unevenly compared to the proposed framework. This scenario slowed block stability convergence and introduced higher variance across blocks compared to the proposed framework, which delayed the initiation of central regression by approximately 36%. This delay propagated to the overall workflow and increased the time required for individual blocks to reach the predefined threshold, leading to an overall increase in end-to-end regression latency. These observations demonstrate that block-level autonomous regression plays a critical role in accelerating stability formation and enabling timely hierarchical progression.

Under the A2 ablation scenario, the user-block bridge agent responsible for hierarchical coordination between user- and block-level regressions was disabled, and the publish-subscribe based coordination was replaced with direct point-to-point interactions among regression agents. In this configuration, we instantiated block regression agents for two representative blocks and user regression agents for three engineers responsible for verifying those blocks. Without a user-block bridge layer, each agent was required to exchange regression status and results through direct communication. However, issuing and receiving direct requests between independently running agents was not well suited to the operational environment. To compensate, a database-backed API was introduced to store regression outcomes and expose them to other agents. Accordingly, block regression agents were forced to periodically poll the API to detect updates from user regression agents rather than reacting to events asynchronously. This polling-based interaction increased coordination latency and introduced unnecessary overhead. Furthermore, when additional user regression agent was added, the API schema and query logic had to be explicitly updated to ensure that block-level agents recognize and aggregate the new results. This tight coupling between agent logic and coordination mechanisms reduced scalability and increased maintenance complexity. Overall, the absence of publish–subscribe coordination eliminated event-driven decoupling and weakened fault isolation while imposing additional integration overhead. This result highlights the importance of hierarchical coordination mechanisms for scalable and maintainable regression.

In the A3 configuration, the regression execution agent responsible for dependency management and execution monitoring was removed and these functions were replaced with static execution scripts. While this configuration preserved the ability to run regressions, execution became tightly coupled to predefined scripts without monitoring, retry handling, or dependency awareness. As a result, failures caused by transient infrastructure issues required manual intervention and execution progress did not be adaptively adjusted based on runtime conditions. In addition, the removal of the regression execution agent eliminated input-driven environment configuration, which previously allowed executions to be dynamically tailored to different verification environments. Under a static script-based setup, supporting such variations required manual adjustment of script options or the invocation of separate scripts for each configuration which increased operational overhead and maintenance complexity. Furthermore, static script-based execution limited parallelism and reduced responsiveness to workload fluctuations, leading to longer execution queues under concurrent regression requests. The absence of centralized execution monitoring also reduced visibility into job status and delayed failure detection. Collectively, these effects increased execution latency and constrained scalability, demonstrating that a dedicated regression execution agent is essential for robust fault handling, efficient resource utilization, and flexible execution across diverse verification environments.

Across the A1–A3 ablation scenarios, the results consistently demonstrate that each agent type contributes a distinct and non-overlapping role to the proposed hierarchical regression framework. Removing the block regression agent (A1) delayed block-level stability convergence and significantly increased the time required to trigger central regression. Disabling the user–block bridge agent (A2) replaced publish-subscribe-based coordination with direct agent interactions, forcing polling-based result exchange through an external API. This eliminated event-driven decoupling, increased coordination latency and integration overhead, ultimately reducing scalability and maintainability of the regression workflow. Replacing the regression execution agent with static scripts (A3) degraded execution robustness and scalability by removing fault handling, and input-driven environment configuration. Collectively, these results confirm that block-level autonomy hierarchical coordination and execution orchestration are all essential for achieving scalable and maintainable regression management. In addition, the findings suggest that other agents provide complementary functionality required for a complete and robust regression framework.

IV. Conclusion

This paper presented a 3-tiered agentic AI regression framework that restructures verification workflows by aligning regression scope with the locality of design changes and verification maturity. The 3-tiered architecture enables verification activities to be performed at the appropriate level of integration with a publish-subscribe backbone,

allowing scalable interoperability and heterogenous verification environments of chip-level regression. The framework also leverages the agentic AI components that are classified into three types to address distinct responsibilities within the verification regression. This separation of responsibilities enables the regression workflows to move away from rigid, script-driven execution toward a structured and scalable framework. As a result, the proposed framework enables the regression becomes more predictable, traceable, and responsive to ongoing design changes, while maintaining deterministic control through explicit gating conditions.

Although large-scale quantitative evaluation is still in progress, early experimental results and deployment observations suggest that the proposed hierarchical gating and agent-driven orchestration significantly reduce redundant regression execution and compress the turn-around-time of verification regression. In particular, enforcing block-level qualification criteria prior to central regression prevents unstable or incomplete verification states from propagating into chip-level, leading to a measurable reduction in failure noise and re-run overhead. As a consequence, a large portion of verification issues are detected and resolved at the user and block regression tiers, allowing central regression to focus primarily on genuine system-level integration behavior rather than block-local instabilities.

The effectiveness of the proposed architecture arises from the coordinated interaction between the 3-tier architecture with the agent role decomposition based on publish-subscribe structure. The tiered organization aligns verification scope with the locality of design changes and verification maturity, clarifying the appropriate level at which verification should be performed. At the same time, the agent role decomposition separates verification decision-making from coordination and execution responsibilities, improving manageability and extensibility across the regression workflow. The publish–subscribe backbone enables asynchronous and fault-tolerant propagation of regression state while avoiding tight coupling between tiers. As a result, the verification regression remains stable under continuous design changes and distributed verification environments, where monolithic scripted regression pipelines become difficult to sustain.

As part of future work, the proposed framework will be extended to multi-project and multi-team environments, where cross-IP dependencies and shared infrastructure further increase the need for controlled regression workflow. Additional effort will focus on strengthening the role of agentic AI within the proposed framework by incorporating richer contextual signals into change analysis and failure classification. These signals include unstructured design discussions, historical debug traces, and previously unused verification artifacts that are currently underutilized in conventional regression workflows. Finally, integration with MCP-based tools will further standardize agent interaction with external systems, enabling more robust tool interoperability and extensibility. These extensions aim not to replace human expertise, but to systematically reduce manual coordination overhead and allow engineers to focus on high-value debugging and verification itself.

In summary, the proposed 3-tiered agentic AI regression framework represents more than a structural reorganization of verification regression workflows. It presents a concrete architectural approach for transforming regression into a managed and autonomous verification process with explicit qualification criteria for central regression enforcement and fault-locality preservation, while sustaining scalability across growing design complexity and verification size. By distributing regression intelligence across specialized agents, the proposed framework enables effective regression governance while providing a robust foundation for integrating AI-driven capabilities into verification workflows.

REFERENCES

[1] Sneha Gokarakonda, et al. "Automated vManager regression using Jenkins"
[2] Lingkai Shi, et al. "Automating Regression Triage in Design Verification Using AI-Based Random Forest Models"
[3] Jin Choi, et al. "A Large Language Model–Based Framework for Enhancing Integrated Regression"