Unified AI-Driven Verification: Combining Spec-RAG, Memory Networks, and Generative AI

Seonghee Yim[1] Insoo Jang[2] Hanna Jang[3] Youngsik Kim[4]

Samsung Electronics, Hwaseong, Korea.
sh.yim; is1226.jang; hn01.jang; ys31.kim@samsung.com

***Abstract*- Maintaining traceability between design specifications, verification plans, and test outcomes in System-on-Chip (SoC) projects is complex due to frequent design iterations and evolving requirements. We propose a practical hybrid AI framework integrating HippoRAG (Hippocampal Retrieval-Augmented Generation), dual-memory RAG (STM/LTM), and LLM–Database reasoning across heterogeneous data sources: structured design and plan metadata in Oracle RDB, and unstructured test results in MongoDB. The framework dynamically adapts to SoC development phases—handling high-change periods dominated by specification evolution and stable phases focused on quality convergence. Evaluation on a representative SoC project demonstrates near-human accuracy traceability accuracy and perform root-cause analysis for reduction in debug time in data-rich contexts. Performance decreases in early stages or sparse datasets with weak relational links, indicating that relational density strongly influences reasoning efficiency. These findings demonstrate a realistic and incremental adoption path for AI-assisted verification analytics in industrial SoC design.**

I. Introduction

Modern SoC development evolves continuously through multiple versions of design specifications, verification plans, and regression environments.

Blocks or IPs are versioned and released asynchronously, while test plans and test codes are continuously updated.

Consequently, requirements-to-verification traceability becomes increasingly difficult to maintain manually, especially when artifacts reside in separate databases and formats.

To address this challenge, we propose a hybrid AI traceability framework combining:

- Oracle RDB for structured requirement, design maturity, and verification plan metadata
- MongoDB for unstructured test logs, coverage, and regression results
- HippoRAG for hierarchical provenance reasoning
- STM/LTM dual-memory RAG for phase-aware traceability
- LLM–SQL/NoSQL reasoning for explainable query and report generation

When designers and verification engineers need various types of data which have been generated and utilized in the development stage of flagship SOC, it is quite difficult to receive accurate data, history information of the data, and related analysis information for the following three reasons. First, the diversity of the structures of Design Block and IP, and second, the lack of management of scenario in an unstandardized development environment utilized by complex and numerous validation data validation units and, finally, by developers scattered across multiple organizations is a major factor.

In this paper, we propose **a Dynamic Hybrid AI verification framework** that combines the inference of SoC by storing and configuring it in RAG, RDB, and NoSQL according to the relationship, frequency, and characteristics of data that occur during each development stage of Generative AI. So, this paper introduces a **Generative AI (LLM) and HippoRAG-based hybrid framework** that provides explainable traceability by combining structured database reasoning with semantic graph traversal. The framework captures Δ-based change information between design, test, and specification artifacts, enabling explainable, cross-version reasoning supported by **Short-Term Memory (STM)** and **Long-Term Memory (LTM)** systems

II. Helpful Hints

Traditional approaches rely on spreadsheets or formal models, making traceability difficult to scale. NLP-based methods rarely address hierarchical hardware structures or integrate RTL integrity and verification results. Unlike previous techniques, RAG is effective for document-based reasoning, and DB+LLM approaches have emerged for structured data. Our work combines generative AI and KG-based RAG, RDB, and NoSQL in a hybrid pipeline, enabling traceability and debugging for both structured and unstructured data. The information on the amount of changes occurring in the overall lifecycle of the verification reviewed in this paper and the database system to be used are as follows.

Design–Verification Lifecycle

Design progresses through two characteristic phases:

1. **High-Change Phase :**
   - Frequent specification updates
   - Rapid design iteration per block/IP
   - Frequent modification of verification plans and test codes
   - Sparse relational data → reasoning dominated by semantic expansion
2. **Mature Phase :**
   - Stable design and plan versions
   - Verification coverage convergence
   - Dense relational data → reasoning dominated by structured SQL/NoSQL precision

Data Base Systems

- **Oracle Database:**
  - Stores DESIGN_INTEGRATION_META_DATA, BLOCK/IP_DESIGN_INFO, VERIFICATION_PLAN, and MATURITY_STATUS tables
  - Each includes structured fields for requirement IDs, maturity stages, plan versions, and release tags.
- **MongoDB:**
  - Stores test_results, coverage_runs, and regression logs.
  - Provides fast access to recent test sessions and real-time sanity data.

The contents below explain the definition of Technical terminology used in the paper and the meaning of contextual use.

Delta-edges (Δ-based change):

Definition: Delta-edges refer to edges in a knowledge graph that represent changes (deltas) in relationships or attributes between nodes over time. These edges capture dynamic updates, such as when a property value is modified, a new dependency is introduced, or a relationship is removed.

Contextual Explanation: A delta-edge is a specialized edge in a knowledge graph that explicitly represents a change (or "delta") in a system's state, such as a modification to a design specification, a hardware component, or a verification plan. It is not just a static link between two nodes, but a temporal and semantic connector that captures the "before" and "after" of a change, along with metadata about the change itself (e.g., who made it, when, and why).

Provenance-Aware Graphs:

Definition: A provenance-aware graph is a knowledge graph that explicitly tracks the origin and history of each piece of information (node or edge), including where it came from, how it was derived, and what processes or data influenced it.

Contextual Explanation: Every generated answer can be audited by tracing back to the original data sources. Hallucinations are reduced, because the model must justify each statement with a verifiable path in the graph.

HippoRAG Concept and Reasoning:

Concept: HippoRAG (Hippocampal Retrieval-Augmented Generation) is a new method that gives large language models (LLMs) a form of long-term memory inspired by the human brain (specifically, the hippocampus). Instead of storing information in isolated chunks, it organizes knowledge into a connected network called a knowledge graph (KG).

Reasoning: The main goal is to improve the AI's ability to answer complex questions that require multi-step reasoning (linking facts across multiple documents). Traditional RAG struggles to connect information across separate document chunks. HippoRAG uses the knowledge graph to efficiently find and integrate all related information in a single, fast step, just as a human brain links associated memories. This makes it faster, cheaper, and more accurate for complex tasks compared to older methods

In AI and cognitive science, the dual-memory layer concept distinguishes between two types of information storage, much like in the human brain: Short-Term Memory (STM) and Long-Term Memory (LTM).

Short-Term Memory (STM)

Concept: This is the AI's "working scratchpad" for the immediate conversation or task. It holds the temporary context needed to respond coherently in the moment.

Characteristics:

- Temporary: Information lasts only for the duration of the current session or a few turns of dialogue.
- Limited Capacity: It's usually managed by the model's fixed "context window" or a conversation buffer. Once this window is full, older information is forgotten.

Long-Term Memory (LTM)

Concept: This acts as the AI's permanent archive, designed to store knowledge, preferences, and historical interactions across multiple sessions.

Characteristics:

- Persistent: Information is stored indefinitely, often in external databases (like vector stores or knowledge graphs), and can be retrieved when needed in future interactions.
- Vast Capacity: It has a seemingly unlimited capacity, allowing the AI to "learn" and personalize interactions over time.

STM helps the AI stay focused on the *current* conversation, while LTM gives it a persistent, growing knowledge base that *persists* across time, enabling continuous learning and personalized interactions.

III. PROPOSED METHOD AND EXPERIMENTAL RESULT

The framework we propose **dynamically adjusts the search strategy weights of the Dual-Memory RAG** according to the characteristics of the output and data created and generated in the SoC development stage. We describe in detail the characteristics according to the development stage of SoC and the proposed weight model, and configure RAG and DB-related systems and tool for each memory type as follows.

A. Dual-Memory RAG systems for SoC development process

- **ML1/ML2 High Changer** (Specification Evolution and Early Validation Features)

Spec, design, verification plan, etc. are frequently changed, so fast test execution and debugging work are actively occurring

STM Weight High (0.7 ~ 0.8)

LTM Weight Medium (0.2 ~ 0.3)

Logic Freshness-Based Discovery First: LLM deduces the most recent unstructured test results and error logs extracted from MongoDB (STM). ex) "Is there an operational error in the A function of the current design?"

- **ML3/ML4/MTO Stabilizer** (Stable phases: Quality Convergence and Deep Debugging)

Minimal changes in functional specifications, completion of validation scenarios for multiple block, complex and chronic bugs and debugging of corner cases

STM Weight Low (0.2 ~ 0.4)

LTM Weight Low (0.6 ~ 0.8)

Logical Relationship Density Based Search First: LLM infers a large proportion of the structural and long-term relationships between designs, plans, and requirements extracted from Oracl RDB (LTM). ex) "What is the root cause of this bug?" When analyzing intermittent bugs that occur in a particular test, the framework searches the LTM to comprehensively compare all the requirements associated with that test, the history of design changes, and similar failure patterns in the past.

- HippoRAG represents an SoC artifact — specification, IP block, verification plan, or test result — while edges encode Δ-edges (version changes), temporal provenance, and causal dependencies.

  Graph Composition

  [SOR/HDD Spec] → [Design Block/IP Version] → [Verification Plan] → [Test Results] → [Issues]

  - Nodes: Design Structure based embedding + metadata (version, owner, spec section)
  - Edges: Δ lineage, revision history, requirement–verification feature or function mapping
  - Query Handling: Both upstream (design spec→ test plan → test result → issue) and downstream (issue → test result → test plan → design spec) traceability

  Benefits

  - Enables explainable and version-aware reasoning
  - Reduces manual effort in building trace matrices

- Integrates with both structured and unstructured data sources

When combined, STM provides immediacy, while LTM ensures historical consistency and explain ability.

B. Methodology and Data Architecture

The framework combines Δ-based knowledge capture, memory-based reasoning, and multimodal database querying.

Oracle RDB Schema Example

The structured artifacts are stored in Oracle.
Key tables include:

Table	Example Columns	Description
PROJECT_OWNER	REQ_ID, REQ_DESC, SPEC_SECTION, VERSION, OWNER_TEAM	Master requirement index.
TEST_PLAN	PLAN_ID, BLOCK_NAME, FEATURE_TAG, TESTCASE_NAME, MATURITY_LEVEL	Verification plan metadata for each block.
DESIGN_BASELINE	RELEASE_TAG, DESIGN_REV, SPEC_VERSION, MATURITY_STAGE	Mapping between design releases and specification versions.
MATURITY_STATUS	BLOCK_NAME, DESIGN_STAGE, APPROVAL_FLAG, TIMESTAMP	Tracks design maturity level and approval status.

Example query generated by the LLM layer:

```sql
SELECT PLAN_ID, FEATURE, SUBFEATURE, OWNER, PRIORITY FROM TEST_PLAN
WHERE BLOCK_NAME = 'DMA' AND DESIGN_VERSION = 'EVT0_ML3_DEV05';
```

MongoDB Collection Example

Collection	Key Fields	Description
test_results	run_id, block, status, timestamp, release_tag	Regression outcome for each test run.
coverage_runs	run_id, block, coverage_rate, feature_tag	Coverage metrics captured per regression.
logs	log_id, test_name, summary, error_trace	Execution logs and error traces for debugging.

Example Query:

```
db.test_results.find({ project: "S5EXXXX", final_result: "FAILED", version: RegExp("EVT0_M
L3_DEV04|EVT0_ML3_DEV03", "i"), test_name: RegExp("(?=.*abox)(?=.*aud)", "i")})
```

C. Results and Discussion

In practice, these artifacts are stored across different repositories (Confluence, Perforce, file servers, Jira, CI systems), making it difficult to reliably trace the cause of failures.
Conventional troubleshooting relies heavily on manual analysis and "tribal knowledge," leading to inconsistent debugging quality and long issue-resolution cycles. While RAG (Retrieval-Augmented Generation) has emerged as a promising technique, naïve RAG approaches do not consider the temporal volatility of SoC data nor the structural relationships among requirements, design, verification assets, and issue histories. To address these issues, this paper introduces a **Dynamic Dual-Memory RAG AI Framework** that integrates:

Components:

STM (MongoDB, Search Engine Hybrid): recent logs, diffs, failure signatures
LTM (Oracle RDB, HippoRAG, Neo4j): requirement–design–test–issue relations
RAG: specification, RTL, verification documentation
Dynamic Weighting Controller(Decision model:MiniMax-M2 detect ML stage & memory weight control): adjusts STM/LTM ratio
LLM Reasoning Engine(Action Model:GLM-4.6): Actual debugging, i.e. STM/LTM evidence and generate root cause description and synthesizes evidence

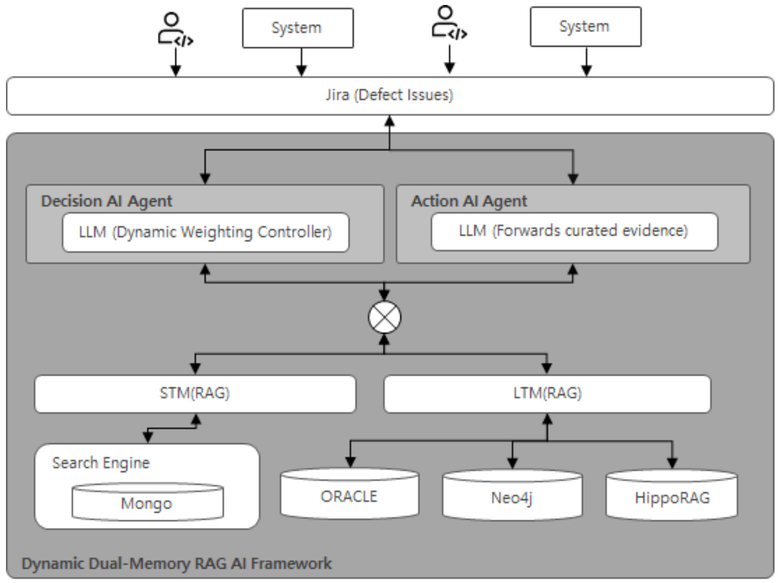

Figure 1 Dynamic Dual-Memory RAG AI Framework

Integrated Pipeline (Figure 1: Overall Agent Orchestration):

1. Decision Model(MiniMax-M2) detect ML state and analyze data type, reason and category of the issue type using Metadata. And Compute the STM/LTM weight based on that information
2. Gathering the evidence information from STM(latest log, latest diff of designs and spec and test fail case pattern) and LTM(spec, design, test plan and issue relation, historical issue reasons)
3. Aggregates and ranks evidence.
4. Forwards all curated evidence to Action Model(GLM-4.6) for deep analysis
5. Returns a final, structured debugging answer and serving

Experimental Results:

This section validates the proposed idea using empirical data from an actual SoC development project. We first describe the data pipeline structure within the proposed PoC framework and then present validation results focused on efficiency gains through real-world data examples.

1. Data Analysis Methodology

To establish a baseline and evaluate the effectiveness of the proposed model, we conducted a static analysis of significant data variations and update cycles. The analyzed datasets include design data (specifications, RTL) and verification data (verification plans, testbenches). Final simulation results are excluded as they are consequential data; instead, we incorporate Jira issue data triggered by these datasets. Jira records provide a detailed action history of defect resolution. By correlating issue reasons with "Time-to-Resolve" and data dependencies, we quantify the impact of our proposed framework.

To facilitate this, we utilize the **RTLCIC (RTL Issue Checker)** system, an internally developed tool. This system manages fixed labels for design defects by tracking "issue-detected versions" versus "resolution plan versions," acting as a gating mechanism during block releases.

2. Efficiency Validation by Error Type

Type 1: Simple Information Mismatch/Missing (60–90%: 75% Avg Efficiency Gain)

The framework effectively identifies false alarms by analyzing provided metadata. When verification engineers report issues based on misunderstood information, the system enables immediate resolution using grounded data. These errors primarily involve specification discrepancies—such as memory maps or SFR settings—and are categorized as "False Alarm" or "Spec-Out" issues.

Type 2: Errors Requiring RTL Modifications (10–50%: 30% Avg Efficiency Gain)

These errors occur as functional failures depending on development maturity and are categorized into Spec Fixed, RTL Issue Fixed, Maturity Issue, and RTL Issue Workaround. The frequency and data lifecycle of these errors vary by development stage.
Case Study (Spec Fixed Type): During chip-level integration of blocks or IPs, errors often arise from incorrect port declarations that deviate from specifications. Furthermore, redundant errors often persist in older versions even after being resolved in subsequent releases. The proposed system reduces debugging time for these cases to near zero.
Implementation Example: By utilizing the RTLCIC system to track target fix versions, we eliminate inefficient repetitive runs across different RTL versions. For instance, a SYSREG connection error in "Block A" during the EVT0 phase was detected in version EVT0_ML2_DEV06 and resolved in EVT0_ML2_DEV09.
Quality Contribution: Data-driven traceability ensures that if no changes occur in the design and no updated test results are present, the issue is flagged as persistent. This eliminates verification "holes" and significantly enhances overall verification quality.

Temporal Analysis: Issue Trend vs. Memory Utilization

Stage	Issue count*(Avg/Week)	Volume Memory Dominance	Rationale
ML1	Highest(16)	STM ↑	frequent diffs, unstable logs
ML2	High(10)	STM > LTM	active feature rollout
ML3	Moderate(6)	LTM ↑	structural & relational issues
ML4	Low(2)	LTM > STM	convergence & verification closure

Efficiency Validation by Error Type:

Stage	Type1 Issue Ratio	Type2 Issue Ratio	Debug Time Reduction
ML1/ML2	34/194	58/194	9~13%
ML3/ML4	73/204	77/204	11~27%

Ablation Summary:
To quantify the contribution of each component, we performed ablation experiments with four variants of the system:

1. Baseline: No dual memory, no dynamic weighting. Keyword search only.
2. Dual Memory (Static): STM + LTM combined, but equal weights (50/50).
3. Dynamic Weighting Only: Memory controller active, but no structured LTM graph.
4. ~~Full System (Proposed): Dynamic STM/LTM weighting + RAG + Oracle + Neo4j graph + GLM reasoning~~ (Future Work)

Final Comparison:

Variant	Debug Time Reduction	Notes
Baseline	0%	keyword/manual search
Static Dual Memory	11%	no dynamic adaptation
Dynamic Weight (HippoRAG + STM/LTM)	27%	no auto issue triage with graph reasoning

The results demonstrate a significant performance degradation when the dynamic strategy is ablated: Impaired Recency in ML1/ML2: In the high-change phases (ML1/ML2), the static RAG performed poorly in resolving false alarms (107 issues). Without prioritizing the "Recency" logic via high STM weights, the system frequently retrieved outdated data, leading to continued debugging efforts on obsolete issues.
Weak Relational Analysis in ML3/ML4: During stable phases (ML3/ML4), the static RAG struggled with complex root cause analysis. The lack of an explicitly high LTM weight strategy prevented the LLM from efficiently leveraging the "Design Relational Density" data in the RDB, resulting in a ~27% performance delta compared to the full system.
Dynamic weighting is critical because different ML stages have fundamentally different information flows.

D. Conclusion and Future Work

This study presents a Dynamic Dual-Memory Retrieval-Augmented Generation (RAG) Framework designed to enhance debugging and verification efficiency in industrial System-on-Chip (SoC) development. The framework integrates two complementary memory systems:

Short-Term Memory (STM): rapidly evolving data such as regression logs, runtime errors, and recent code diffs

Long-Term Memory (LTM): stable and relational information, including requirement–spec–RTL–test mappings and historical design context

A decision model (MiniMax-M2) analyzes each user query to determine the SoC development stage (ML1–ML4) and the associated data-change characteristics. Using this information, the Memory Controller dynamically adjusts STM/LTM retrieval weights, enabling the system to adapt to the shifting data landscape of SoC development. The action model (GLM-4.6) then performs reasoning using the curated evidence.

Our evaluation uses 34 weeks of real SoC development data, including 398 issues across multiple categories such as connection, function, false-spec, and testbench failures. Experimental results based solely on historical data show that the proposed system:

Reduces estimated debugging time up to 27%. Provides the greatest improvements in false-spec, connection, function, and testbench issue categories. Enhances relevance and quality of retrieved evidence, especially in early-stage (ML1–ML2) volatile environments and later-stage (ML3–ML4) structural debugging

Ablation results confirm that key components—dual-memory separation, dynamic weighting, and LLM-driven decision logic—each contribute significantly to overall gains. In summary, the proposed framework effectively captures the distinct information needs of different SoC development stages. By combining adaptive memory utilization with LLM-based reasoning, it delivers a practical AI-assisted verification approach that meaningfully accelerates debugging workflows using only available historical data, without relying on timing or CDC analysis or error-type classification.

Future Work

1. Integration with Real-Time Verification Pipelines

Future extensions will integrate the framework into continuous verification workflows, enabling real-time ingestion of logs, diffs, and execution traces. This will allow the system to proactively detect anomalies and suggest debugging actions even before engineers begin manual analysis.

2. Enhanced Graph-Based Reasoning

The current Neo4j-based LTM enables multi-hop dependency reasoning; however, future research may explore graph neural networks (GNNs) or graph transformers to automatically infer latent architectural relationships and failure propagation paths, improving system performance for complex corner-case issues.

3. Adaptive Learning via Human–AI Feedback Loops

As verification engineers interact with the system, feedback signals can be used to refine retrieval ranking, STM/LTM weighting, and decision-model accuracy. User-specific tuning may yield further improvements in debugging speed and relevance.

4. Automated Issue-Type Classification to Replace Human Label Noise

A significant limitation of the current dataset is that issue types were originally labeled manually by engineers. Such human-labeled data often contain inconsistencies, category drift, or misclassification—particularly in ambiguous categories such as “uncategorized,” “misc.” or “false-spec.”

Future work will incorporate an AI-driven issue classification module that automatically:

- Infers issue categories from logs, diffs, and test outcomes,
- Corrects mislabeled historical data using self-supervised refinement,
- Enforces consistency between issue reason, module context, and ML stage,
- Reclassifies ambiguous or mixed-type issues using multi-label inference.

By replacing noisy human-labeled data with consistent AI-driven labels, the system will not only improve the decision model’s reliability but also refine STM/LTM weighting, similarity retrieval accuracy, and performance prediction across development stages. This enhancement is expected to substantially improve robustness, especially in ML1 and ML3 where classification ambiguity is historically highest.

References

[1] Bernal Jiménez Gutiérrez, Yiheng Shu, Yu Gu, Michihiro Yasunaga, Yu Su, "HippoRAG: Neurobiologically Inspired Long-Term Memory for Large Language Models" NeurIPS 2024

[2] Zihong He, Weizhe Lin, Hao Zheng, Fan Zhang, Matt W. Jones, Laurence Aitchison, Xuhai Xu, Miao Liu, Per Ola Kristensson, Junxiao Shen1, X-Intelligence Labs, 2 University of Cambridge, 3 University of Bristol, 4 Columbia University, 5 Meta, "Human-inspired Perspectives: A Survey on AI Long-term Memory" arXiv 2411.00489

[3] Seonghee Yim, Hanna, Sunchang Choi and Seonil Brian Choi, "Accelerating SOC Verification using Process Automation and Integration" Design Verification Conference (DVCon), 2020