AGREE-Dog Copilot: A Neuro-Symbolic Approach to Enhanced Model-Based Systems Engineering

Amer Tahat, Isaac Aumundson, David Hardin, and Darren Cofer

Collins Aerospace
Cedar Rapids, IA 52498 USA
{amer.tahat, isaac.aumndson, david.hardin, daren.cofer}@collins.com

Abstract. Formal verification tools like model checkers have long demonstrated their capability to ensure mission-critical properties are satisfied, yet their adoption in the aerospace and defense industries remains limited. Surveys consistently identify difficulty in interpreting analysis results, especially counterexamples, as a primary barrier. Previously, our team developed AGREE, an assume-guarantee compositional reasoning tool for architectural models, which generates detailed but often challenging-to-interpret counterexamples. In this paper, we introduce AGREE-Dog, an open-source generative AI copilot integrated into the OSATE IDE to enhance explainable compositional reasoning with AGREE and AADL. AGREE-Dog automates 16 DevOps and ProofOps steps, utilizing a novel context-selection and memory management system to efficiently manage evolving artifacts and historical interactions. We introduce structural and temporal metrics to evaluate the typically overlooked human contributions in generative AI-supported workflows. Evaluations using 13 UV fault-injection scenarios demonstrate a significant reduction in manual effort (less than 0.1 % of tokens authored by users), rapid convergence of counterexample repairs (84.6 % resolved in a single iteration, accuracy increasing to about 92 % after two iterations, and reaching 100 % within three iterations), and low latency (average LLM response under 22 seconds, with negligible AGREE-Dog computational overhead). We also discuss limitations and future work. These promising results motivate further exploration into explainable model-based systems engineering (MBSE).

Keywords: LLM · Formal verification · MBSE · AGREE · AADL · Compositional reasoning

1 Introduction

Formal methods provide a mathematically rigorous means of verifying correctness in high-assurance systems, such as those used in the aerospace and defense industries. Certification guidance such as DO-333 [13] explicitly outlines how formal methods can meet airworthiness objectives for commercial aircraft

software. Despite their proven effectiveness, adoption within traditional development workflows remains limited, hampered by scalability challenges, poorly designed tooling, and significant barriers to entry due to specialized training requirements [3].

The DARPA Pipelined Reasoning of Verifiers Enabling Robust Systems (PROVERS) program was launched to address these adoption barriers by developing scalable, human-centered formal verification workflows that seamlessly integrate into existing aerospace and defense engineering practices. Central to PROVERS' objectives is enabling usability even among engineers who lack extensive formal methods expertise, thereby fostering broader adoption and enhancing system dependability.

In response, our team has developed the Industrial-Scale Proof Engineering for Critical Trustworthy Applications (INSPECTA) framework [6]. INSPECTA comprises two integrated layers—*ProofOps* and *DevOps*—that embed formal verification directly into modern DevOps pipelines. The framework emphasizes scalability and explainability as primary design objectives, aligning closely with the PROVERS program's goals.

Within INSPECTA's *ProofOps* workflow, we employ the Assume-Guarantee Reasoning Environment (AGREE) [2], a compositional verification tool designed specifically for the Architecture Analysis and Design Language (AADL) [4]. Although AGREE avoids many of the scalability pitfalls found in monolithic verification tools, its counterexample outputs remain difficult to interpret. Like many model checkers, AGREE produces tabular counterexamples that trace the state of variables across multiple time steps. These can involve intricate temporal logic, nested states, and violations spanning architectural layers, posing challenges even for experienced engineers [7]. The diagnostic and repair process may span, hours, days, or weeks for large, evolving models based on user experties.

Recently, generative AI, and particularly large language models (LLMs), have shown promising potential to improve explainability and guide automated formal verification and counterexample repair. Early efforts include OpenAI's GPT-f, which achieved notable success in Metamath theorem proving [8, 12]. Other initiatives have applied LLMs successfully to proof repair in Isabelle/HOL [5], theorem diagnosis in Coq [18], and discovering program invariants [11, 17]. Stanford and VMware's Clover project represents another significant step forward, focusing on verifiable code generation with generative assistance [14]. Tahat et al. demonstrated high success rates using multi-turn conversational LLMs for proof repair in Coq, underscoring conversational learning's value in formal reasoning domains [15, 16]. Apple's GSM-Symbolic [9] highlighted fundamental limitations of LLMs in symbolic reasoning tasks. Similarly, Amazon's recent SMT-backed hallucination prevention framework [1], while innovative, remains closed-source, available exclusively as a web service, and has yet to integrate within aerospace-specific MBSE pipelines such as those based on AADL.

In this paper, we introduce AGREE-Dog, an open-source generative AI copilot integrated directly into the OSATE IDE. AGREE-Dog is purpose-built to support explainable, compositional reasoning within MBSE workflows, partic-

ularly targeting aerospace and defense applications using AADL and AGREE. It automates 16 DevOps and ProofOps steps, including requirements ingestion, context-aware prompt construction, semantic diffing, formal validation, and log analysis. Notably, AGREE-Dog employs a specialized conversational memory management and context-selection approach that efficiently represents evolving verification artifacts in a concise, fixed-size token window, dramatically simplifying the complexity inherent in tracking and repairing system-level contract violations (see Section 6 for detailed evaluation and artifact descriptions).

To quantify these benefits, we introduce novel structural and temporal evaluation metrics, explicitly addressing aspects such as repair convergence, human intervention level, token efficiency, and response latency.

The rest of this paper is organized as follows. Section 4 presents AGREE-Dog's architecture and orchestration strategy. Section 6 details our experimental evaluation and findings. We conclude with current limitations and outline future work aimed at enhancing autonomy and generalization across verification domains.

2 Explainable AGREE

2.1 Overview

AGREE provides a formal contract language for specifying *assumptions* (i.e., expectations on a component's input and the environment) and *guarantees* (i.e., bounds on a component's behavior). Because AGREE is implemented as an AADL *annex* in the Open Source AADL Tool Environment (OSATE), the contracts are specified directly on components in the AADL model. AGREE then uses a k-induction model checker to prove properties about one layer of the architecture using properties allocated to subcomponents. The analysis proves correctness of (1) component interfaces, such that the output guarantees of each component must be strong enough to satisfy the input assumptions of downstream components, and (2) component implementations, such that the input assumptions of a system along with the output guarantees of its sub-components must be strong enough to satisfy its output guarantees.

When a contract violation is found (i.e., when an assumption is determined to be invalid or a guarantee is unsupported), AGREE produces a counterexample consisting of values for each system variable at each execution step. A sample counterexample is depicted in Figure 1. Currently, OSATE includes the AADL Simulator tool that can accept an AGREE counterexample as input and walk through the trace in the graphical editor, but it is of limited help when it comes to identifying the root cause of the contract violation.

2.2 Making Counterexamples Actionable

We therefore desire AGREE counterexamples that are *actionable*; that is, an explanation of the violation in terms that will quickly lead to a passing analysis

Counterexample

Variables for the selected component implementation

Variable Name	0	1	2
Inputs:			
{Target Speed.val}	121	0	0
{Target Tire Pitch.val}	0	1/5	0
State:			
{[G car 1] actual speed is less than constant target speed}	true	true	false
{_TOP.AXL..ASSUME.HIST}	true	true	true
{_TOP.CNTRL..ASSUME.HIST}	true	true	true
{_TOP.SM..ASSUME.HIST}	true	true	true
{_TOP.THROT..ASSUME.HIST}	true	true	true
{const tar speed}	true	false	true
Outputs:			
{Actual Speed.val}	11	10	100/11
{Actual Tire Pitch.val}	0	1/5	0
{State Signal.val}	0	0	0

Variables for AXL

Variable Name	0	1	2
Inputs:			
{AXL.Speed.val}	46	46	46
{AXL.Target Tire Direction.val}	0	1/5	0
State:			
{AXL..ASSUME.HIST}	true	true	true
Outputs:			
{AXL.Actual Tire Direction.val}	0	1/5	0

Fig. 1: AGREE counterexample generated from the `Car` model.

(e.g., by making changes to the formal contract or model). To achieve this, we implemented an interactive conversational copilot (AGREE-Dog) powered by GPT-4o and O3 multimodal generative AI models. It is specifically designed to assist AGREE users in identifying the root causes of counterexamples and applying targeted modifications during the model repair process, significantly reducing the turnaround time between verification attempts. The copilot is user-friendly and integrates seamlessly with the OSATE IDE (see Figure 2).

In the remainder of this paper, we explore the motivations that drove the development of AGREE-Dog, describe its key architectural features, and evaluate its effectiveness within representative modeling and verification workflows.

3 Motivations and Core Challenges

Drawing upon our practical experience integrating AGREE within MBSE workflows, in this section we highlight central challenges and key design principles that guided the development of our LLM-based solution for generating actionable counterexample explanations and facilitating automated model repairs.

3.1 Context-Aware Prompt Construction

AGREE-generated counterexamples typically involve numerous variables, intricate execution traces, and extensive AADL architectural data. Incorporating detailed LLM-generated code explanations and diagnostics exacerbates this challenge. Presenting these details directly to a generative AI model without careful

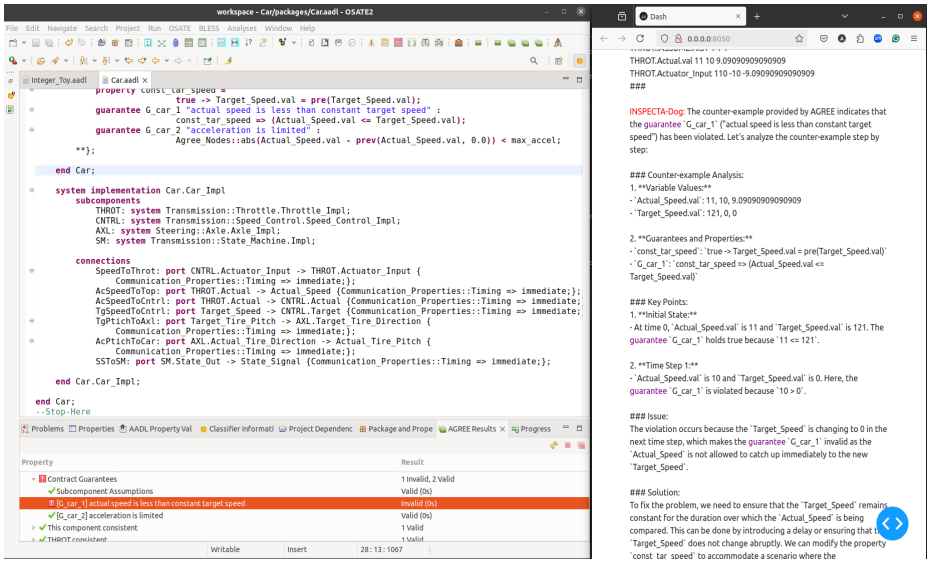


Fig. 2: AGREE-Dog copilot integrated within OSATE, providing an actionable explanation of a counterexample generated from the `Car` model.

management often result in excessive context size, increasing latency, hallucinations, costs, and potentially exceeding token limits. The key challenge is identifying and selecting only the most relevant context to include in prompts, ensuring accurate, concise explanations and actionable recommendations.

3.2 Ensuring Validity of Automated Repairs

Generative models might propose repairs that, while plausible, could unintentionally violate established architectural interfaces or critical system properties. Maintaining consistency within compositional reasoning frameworks, such as AGREE, requires continuous validation. Thus, repairs must be tightly integrated with formal verification steps to ensure that each modification preserves overall system correctness.

3.3 Minimizing User Effort and Interaction Latency

Manually reviewing detailed logs and deeply nested temporal logic from counterexamples is both error-prone and time-consuming. An effective repair process must significantly reduce user overhead by automating log analysis, semantic comparisons between successive runs, and managing formal proof re-validation. Minimizing both system latency and human interaction time is essential to achieve an efficient, near-interactive model repair workflow.

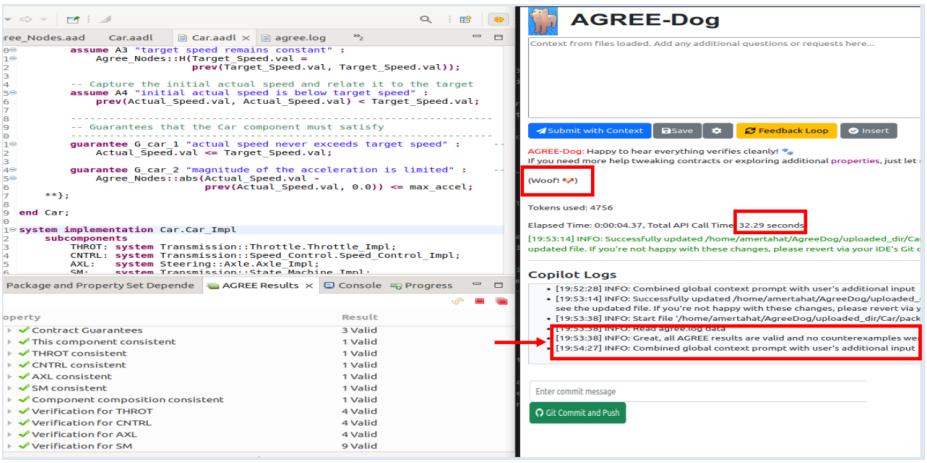


Fig. 3: AGREE-Dog UI interface showing integrated model diagnostics, user input, token count, response time, and push-button feedback loop. Each repair cycle is proof-aware and synchronized with AGREE log results.

4 AGREE-Dog Architecture

This section details AGREE-Dog's architecture (Figure 4), systematically highlighting key subsystems designed to overcome the practical challenges identified previously (Section 3). Specifically, in this section, we present AGREE-Dog's intuitive user interface, sophisticated memory and context management algorithms, formal validation-driven feedback loops, and integration with OSATE and version control systems.

4.1 User Interface and Interaction Workflow

AGREE-Dog features an intuitive, streamlined user interface (UI), (Figure 3), seamlessly integrated within the OSATE environment, designed specifically to minimize cognitive load and simplify complex verification tasks. Central to its usability are clearly labeled, push-button controls, enabling users to directly interact with counterexample explanations, formal validations, and system-level model repairs from a single coherent point of interaction.

A fundamental design principle of this UI is to balance transparency with abstraction—clearly presenting operational outcomes without burdening users with underlying complexities. This approach promotes efficiency, productivity, and verification effectiveness.

At the center of user interaction is the *Feedback* button, which synchronizes the internal state of OSATE with AGREE-Dog, updating its variables and internal data structures. This synchronization ensures coherence between AGREE-Dog's conversational state and the current OSATE project status, thus setting

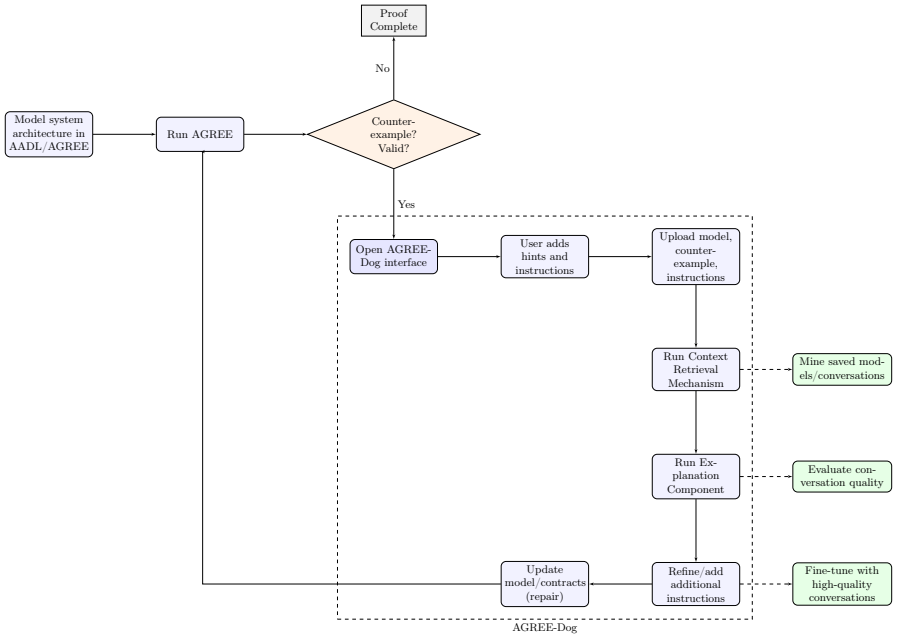


Fig. 4: AGREE-Dog workflow illustrating the integration of formal verification with context-aware explanation and iterative model repair.

the stage for effective model analysis and refinement—detailed further in the next sections.

We complement this mechanism, with the *Insert* button which enables seamless integration of AGREE-Dog's suggested model repairs directly into OSATE, significantly streamlining what would otherwise be a tedious manual integration process. User-driven requests or specific instructions are submitted via the *Submit* button and can be further elaborated upon through an integrated conversational chat window. This conversational approach encourages precise, targeted refinements by enabling iterative and detailed guidance from the user.

Additional UI elements enhance interaction quality and knowledge retention. The *Save* button allows users to archive conversational histories for later review or further analysis and evaluations, as shown in Section 6, while the integrated *Git* control provides mechanisms for persistent storage, sharing of verification outcomes, and collaborative insight generation.

Moreover, advanced configurations are accessible via the dedicated *Settings* menu, allowing users to customize interaction workflows and select optimal LLM models tailored to specific tasks—such as generating explanations and repair suggestions (best supported by GPT-O3), or facilitating rapid implementation of code repairs (ideally powered by GPT-4o), as further detailed in Section 4.2.

4.2 Backend Function Call Graph and Workflow Automation

To support interactive workflows, AGREE-Dog automates 16 critical DevOps and ProofOps steps. The backend orchestration, summarized in Figure 7, manages operations ranging from artifact selection and prompt construction to automated AGREE invocations. AGREE-Dog utilizes context and history-aware agents that dynamically select relevant artifacts, perform semantic diffs, and invoke proof engines. Each backend operation is highly optimized, incurring negligible runtime overhead (less than one second per operation), as demonstrated by the empirical results in Section 6.

4.3 Context Selection and Memory Management Optimization

Effective context selection and memory management are critical to AGREE-Dog's ability to provide precise explanations and actionable repairs involving complex AADL artifacts, execution traces, and user instructions. Addressing these challenges requires the sophisticated, carefully optimized mechanisms embedded within AGREE-Dog's core copilot algorithm.

Core Copilot Algorithm Algorithm 1 embodies the central context management strategy of AGREE-Dog, as conceptually outlined in Figure 4. This algorithm integrates intelligent conversational state tracking, dynamic artifact selection, and optimized memory management processes to efficiently support model verification and repair tasks.

Optimized Dynamic Context Retrieval and Updates. **Algorithm 1** dynamically selects a minimal yet sufficient context—including relevant AADL source files, counterexamples, AGREE logs, and system requirements, and interactive user instructions—for accurate verification and effective repair interactions. Leveraging its integrated dynamic Context Retrieval component, the algorithm selectively imports only the most recently updated model artifacts, identified through AGREE-log updates received from OSATE, by traversing dependency chains and referencing stored conversational data.

By default, the context retrieval strategy excludes standard training data such as core libraries typically present in LLM training sets, thus optimizing token usage. However, users retain flexibility to explicitly include or exclude any files from the complete import chain during initialization, incorporating selected context elements into the initial prompt. Once included, these explicitly imported files remain static in memory unless updated explicitly by the user or signaled via AGREE logs. Additionally, natural-language requirement files (e.g., CSV-based inputs), not tracked by AGREE logs, are monitored independently with automatic checks performed every two seconds to detect changes.

This nuero-sympolic (intersymbolic) and user-customizable selection process significantly reduces redundancy, enhances convergence speed toward correct model solutions, minimizes generative model latency, and mitigates hallucinations caused by irrelevant context.

Algorithm 1: AGREE-Dog Interactive Copilot Prompt Construction and Counterexample Handling

Input: AADL Model Files, Counterexample File (optional), System Requirements (optional)
Output: Prompt for GPT-based AGREE-Dog Copilot, Actionable Repair Suggestions
Initialization:
Load command-line arguments: working directory, start file, counterexample, requirements file;
Load OpenAI API key;
Initialize logging system;
Main Procedure:
if *requirement file provided* **then**
    Load and include requirements in prompt context;
else
    Set requirements context to *"No sys_ requirement file provided"*;
Prompt Construction:
Read top-level AADL file from provided workspace;
Parse import chain and extract relevant AADL files (avoid standard libraries);
if *counterexample provided (CLI or file)* **then**
    Load counterexample into context;
else
    Search for recent counterexamples:
    - Check command-line provided counterexample path first.
    - If unavailable, parse `agree.log` for failing contracts.
    - Match failing contracts with available counterexample XML/text files.
    - Extract and format counterexample(s) for inclusion.

Construct comprehensive prompt with:

1. System Requirements (if available)
2. AADL Model Content
3. Counterexample(s) Explanation
4. Explicit instructions for GPT (repair suggestions within AADL syntax)

Interaction and Feedback Loop (via Dash UI):
while *copilot session active* **do**
    Receive additional user input (optional);
    Combine with the current prompt context (if any);
    Submit prompt to GPT-4o/GPT model via OpenAI API;
    Retrieve response:

    - Explain verification failures clearly
    - Suggest repairs in AADL syntax, respecting requirements

    Present GPT response to user;
    Log interaction and update metrics (latency, tokens used, etc.);
    if *user applies modifications* **then**
        Extract AADL repair suggestions from GPT response;
        Safely overwrite the original AADL model file;
        Notify user of successful update or handle exceptions;
Quality Assessment and Logging:
Automatically record metrics (timestamps, token use, latency);
Store interaction logs for future analysis and fine-tuning;
Shutdown Procedure:
On user request, terminate the copilot session gracefully;

Memory Management Optimization Mechanism. A critical component of Algorithm 1 is its internal conversational memory management subsystem, detailed fully in Appendix A. This subsystem employs a structured, list-based representation to balance immediate responsiveness with longer-term conversational persistence. Short-term interactions are retained in readily accessible memory for efficient prompt updates, while less immediate interactions can optionally be saved locally by the user or systematically migrated into persistent storage managed by integrated Git version control. This approach allows AGREE-Dog to effectively recall prior repair strategies and interaction histories, thus enhancing iterative repairs and significantly reducing the overhead associated with manual snapshots management.

Furthermore, AGREE-Dog's memory management strategy directly facilitates ongoing system refinement. Archived conversational histories and validated repairs can subsequently be leveraged to fine-tune the underlying generative models, enabling continual improvement in the quality of explanations and repair suggestions.

4.4 Verification-Aware Feedback Loop and Repair Validity

AGREE-Dog's neuro-symbolic reasoning, achieved by combining AGREE's formal verification with generative AI explanations, establishes a rigorous, verification-aware repair loop. Central to this process, AGREE-Dog invokes AGREE externally via API calls to ensure that all proposed repairs strictly adhere to system-wide consistency and soundness criteria.

This verification-integrated approach not only acts as a safeguard against unsound or logically inconsistent model modifications but also enhances the quality of data fed into the generative model. By proactively filtering invalid suggestions, AGREE-Dog reduces the overall token volume required, thereby significantly improving LLM latency and maintaining model reliability and trustworthiness. Such integration distinctly differentiates AGREE-Dog from purely neural LLM approaches, which inherently lack logical soundness checks and may erroneously group logically distinct, yet superficially similar elements [9, 15, 16].

Additionally, the semantic diffing mechanism embedded in AGREE-Dog detects relevant model changes precisely across iterative repair cycles, facilitating faster convergence to formally valid solutions. This integrated neuro-symbolic loop thus effectively bridges generative AI capabilities with rigorous MBSE based formal verification.

4.5 Traceability, Logging, and Continuous Refinement

The extensive logging within AGREE-Dog serves dual purposes. First, it facilitates real-time diagnostics, enabling rapid identification of effective conversational interactions and successful repair strategies. As illustrated in the AGREE-Dog user interface (Figure 3), key performance indicators—including AGREE validity status, token count, system and human return time, and LLM latency—are

prominently displayed, providing users immediate feedback to gauge interaction effectiveness.

Second, the detailed logs support ongoing system refinement by highlighting conversational patterns consistently associated with high-quality, formally valid repairs. This capability directly informs the metrics employed for evaluating AGREE-Dog's performance, as further detailed in Section 6 and Section 5. By analyzing logged interaction timelines and human response metrics, AGREE-Dog identifies optimal repair strategies, promotes knowledge reuse, and reduces manual intervention, significantly enhancing both short-term repair efficiency and long-term knowledge retention.

5 Evaluation Metrics

This section introduces the core metrics used to evaluate AGREE-Dog's performance. We organize them into two complementary categories: structural (or spatial) metrics, which quantify the shape and volume of interaction, and temporal metrics, which capture responsiveness and turnaround time. Together, these metrics enable a holistic assessment of automation, effort, and cost.

5.1 Structural Metrics

Structural metrics quantify how the repair process unfolds—how many interactions occurred, how much human input was required, and how much computational effort was expended.

Total Token Count (TTC). This metric captures the total number of tokens exchanged during a repair conversation, including both human-authored tokens and those generated by AGREE-Dog—either by the LLM or by the system's prompt constructor:

$$\text{TTC} = \text{Human Tokens} + \text{AGREE-Dog System Tokens} \tag{1}$$

TTC serves as a proxy for computational and financial cost (e.g., token-based billing), independent of who authored the tokens. However, it does not by itself distinguish the extent of human involvement.

Human Input Ratio (HpR). This metric measures the proportion of human-authored tokens relative to the total token count:

$$\text{HpR} = \frac{\text{Human-Authored Tokens}}{\text{Total Tokens in Conversation}} \tag{2}$$

A lower HpR suggests higher automation, with the system contributing more heavily to the conversation. When considered with TTC, this helps differentiate brief, efficient sessions from those with more human effort or verbosity.

Number of Repair Cycles (N_{RC}). This metric counts the number of conversational cycles required to reach a valid system state:

$$N_{\mathrm{RC}} = \text{Number of Repair Cycles Until Validity} \tag{3}$$

Each cycle begins with a `start_file_read` message and ends with a `validity_status: valid` confirmation. Together, N_{RC}, HpR, and TTC form a triplet that reflects the intensity, automation level, and computational cost of the repair process.

Repair Success Rate (RSR). This metric measures how often AGREE-Dog succeeds in exactly N_{RC} cycles:

$$\mathrm{RSR}(N_{\mathrm{RC}}) = \frac{\text{Number of Tests Solved in } N_{\mathrm{RC}} \text{ Cycles}}{\text{Total Number of Tests}} \tag{4}$$

Cumulative Repair Success Rate *(RSR_{acc}).* This cumulative variant captures the percentage of tests solved within a given number of cycles:

$$\mathrm{RSR}_{\mathrm{acc}}(N_{\mathrm{RC}}) = \frac{\text{Number of Tests Solved in } \leq N_{\mathrm{RC}} \text{ Cycles}}{\text{Total Number of Tests}} \tag{5}$$

These success rate metrics extend the basic structural measures to account for convergence and consistency. They are operationalized in Section 6, where we analyze repair outcomes and cycle distributions (see Figure 6a).

5.2 Temporal Metrics

While structural metrics describe what happened during the interaction, temporal metrics quantify how long it took—enabling assessments grounded in real-world engineering effort and user experience.

Wall-Clock Time (WCT) The total elapsed time from the first user input to final validation. WCT serves as a practical proxy for engineering effort and turnaround time. Shorter durations may reflect both efficient execution and the usefulness of AGREE-Dog's guidance.

WCT also conveys a notion of ***Repair Speed***—how many valid tasks are completed per unit of time. For example, in our evaluation (Section 6), the mean WCT per valid cycle was 2 minutes and 9 seconds, with a median of 1 minute and 39 seconds.

LLM Latency The average LLM response time per repair cycle. Lower latency improves interactivity and helps maintain user focus, especially in iterative or multi-step sessions.

Next, we define a dependent metric based on the previous temporal measurements to estimate the human return time.

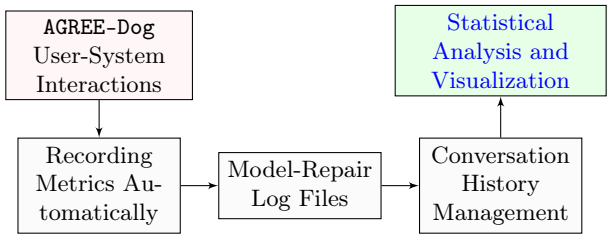


Fig. 5: AGREE-Dog Conversation Quality Assessment Workflow (CQAW). The workflow tracks structural metrics from conversation histories and temporal metrics from copilot logs, leveraging timestamps to measure user and LLM response latencies. Finally, metrics are analyzed and visualized using AGREE-Dog's statistical utility

Human Return Time (HRT) This metric estimates the time required for a human to return to a task and make cognitively informed decisions necessary to reach validity during the interaction. It is calculated as the total wall-clock time minus the time AGREE-Dog spends in CPU execution and large language model (LLM) processing. Formally:

$$\text{HRT} = \text{Wall-Clock Time} - \text{CPU Time} - \text{LLM Response Time} \tag{6}$$

5.3 Composite Score: Structural and Temporal Dimensions

To facilitate comprehensive evaluation, we interpret AGREE-Dog's performance using a composite score that integrates both structural and temporal dimensions:

$$(N_{\text{RC}}, \text{HpR}, \text{TTC}, \text{Wall-Clock Time}, \text{LLM Response Latency}, \text{CPU Time}) \tag{7}$$

This composite vector captures not only the automation level and conciseness of each repair session but also temporal efficiency. For instance, sessions with identical token counts and automation levels might still differ significantly in usability due to variations in latency or total duration. Additionally, this formulation supports the calculation of derived metrics, such as Human Return Time (HRT) (Eq. 6) and Repair Success Rate (RSR).

By combining structural and temporal perspectives, the composite score provides nuanced insights into human–system interaction dynamics, balancing token efficiency with practical engineering outcomes.

6 Experimental Evaluation

6.1 Evaluation Setup and Fault Injection Protocol

Using the Conversation Quality Assessment Workflow (CQAW, Figure 5), we systematically tracked structural and temporal metrics to comprehensively eval-

uate AGREE-Dog. Our experiments involved thirteen fault-injected test scenarios based on an AADL-based Car model. Each scenario featured dynamically evolving artifacts—including AADL source files, natural-language requirements, counterexample traces, AGREE log files, and LLM-generated diagnostics—culminating in approximately 32,100 tokens across all scenarios. On average, scenarios began with around 400 lines of AADL and log content, fewer than 100 lines of counterexample traces, and less than 100 lines of natural-language inputs.

Faults targeted three safety-critical subsystems (Top-Level Control, Steering, and Transmission), triggering 16 repair cycles. Injected faults covered typical behavioral and contract-level violations—ranging from incorrect assumptions, logic errors, and range violations to faulty assignments and temporal inconsistencies. Repairs were accepted only after passing AGREE's formal verification and manual user confirmation via AGREE-Dog's `insert` command, ensuring both correctness and soundness. Detailed scenarios and artifacts are documented in Appendix A and made available in our public GitHub repository[1].

Evaluation Metrics. Figure 6a visualizes repair convergence across the scenarios. Table 6 summarizes AGREE-Dog's structural and temporal performance metrics (defined in Section 5). Next, we summarize the key insights obtained from our evaluation, supported by quantitative data presented in Table 6 and visualized in Figure 6a.

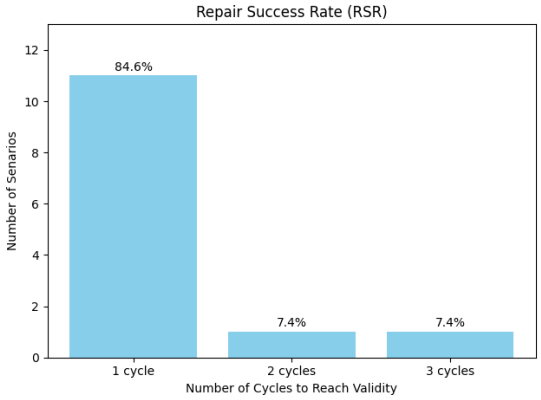
6.2 Key Results.

This evaluation demonstrates the feasibility of integrating generative AI (GenAI) with formal verification in Model-Based Systems Engineering (MBSE). By combining large language model reasoning with AGREE-based validation in OSATE 2, AGREE-Dog delivers verifiable repairs with minimal human effort.

1. **Rapid Convergence with Reduced Human Intervention Frequency:** AGREE-Dog resolved approximately 85% (11 out of 13) of the test cases within a single cycle, while the remaining cases required two or three cycles (approximately 7.5% each). This demonstrates swift convergence and significantly reduces the frequency of user interventions needed across diverse fault scenarios.
2. **High Automation with Minimal Human Effort:** Estiamted by (HpR) metric, Human-generated content constituted less than 0.1 % of the overall tokens, with AGREE-Dog autonomously generating more than 99.9 % via its integrated prompt construction mechanism and language model. Combined with the rapid convergence rate noted previously, this outcome highlights AGREE-Dog's capability to effectively automate model repairs, significantly reducing manual input relative to the extensive verification contexts encountered.

[1] `https://github.com/loonwerks/AgreeDog`

Fig. 6: AGREE-Dog Evaluation Metrics and Convergence



(a) Repair cycles required to achieve system-wide validity.

(b) Structural and Temporal Metrics Summary

Metric	**Result**
Structural Metrics	
System Validity	100% achieved for all test scenarios
Repair Success Rate (RSR)	11/13 (84.6%) in 1 cycle; 1/13 in 2 cycles; 1/13 in 3 cycles
Human Input Ratio (HpR)	$< 0.1\%$ of total tokens
AGREE-Dog Generated Input	$> 99.9\%$ of total tokens
Token Use (per test suite)	4.8k, 5.5k, 22k tokens
Temporal Metrics	
Wall-Clock Time (WCT)	Mean: 2:09 min; Median: 1:39 min
LLM Latency (per cycle)	Mean: 22 s; Range: 4–33 s

3. **Efficiency and Reduced Human Return Time (HRT):**
   AGREE-Dog demonstrated significant computational and cognitive efficiency throughout the evaluation. Internal computational overhead consistently remained below one second per operation, complementing an average LLM latency of approximately 22 seconds per cycle. While the median overall wall-clock time (WCT) was about 1 minute and 39 s the average human response time (HRT) was approximately 1 minute and 3 seconds. This average, however, was notably skewed by two outlier cases; in fact, 85 % of scenarios achieved total resolution (WCT) in under 45 seconds—including LLM latency—limiting human analysis and decision-making time to less than 23 seconds per scenario in 11 out of 13 cases. Compared to traditional manual verification approaches, which typically require hours or days, AGREE-Dog's structured guidance and intuitive natural-language explanations significantly

reduced human cognitive effort estimated by (HRT) metric and the overall interaction duration (WCT).

7 Conclusions and Future Work

To enhance the explainability and usability of AGREE-generated counterexamples, we developed AGREE-Dog, the first open-source conversational copilot specifically integrating neuro-symbolic methods with AGREE's formal verification tools within the OSATE environment. AGREE-Dog produces intuitive, natural-language explanations for complex counterexamples, significantly reducing human effort and cognitive load required for formal model repairs. Our experimental evaluation demonstrates AGREE-Dog's feasibility and effectiveness at realistic MBSE scales—handling scenarios spanning tens of thousands of tokens without notable performance degradation. These initial results provide a promising evidence for the practical utility and scalability of neuro-symbolic methods, highlighting significant potential for broader educational and industrial adoption. AGREE-Dog is publicly accessible on GitHub.

Despite these encouraging outcomes, several avenues for future improvement and exploration remain. We intend to continue evaluating AGREE-Dog on increasingly sophisticated and complex system models and formal specifications.

Furthermore, ongoing developments in large-context language models (e.g., GPT-4.1's 1-million-token context window) offer substantial opportunities to explore more autonomous decision-making frameworks, including reinforcement learning-driven judge-router-worker agentic architectures. Such systems could dynamically and autonomously select optimal repair strategies, further reducing manual intervention. Additionally, extending AGREE-Dog's capabilities to emerging modeling standards, such as SysML v2 [10], represents a key future goal, especially considering that no current SysML v2 tools support comprehensive compositional reasoning or temporal logic analysis comparable to AGREE.

Lastly, the integration of our evaluation workflow into INSPECTA's DevOps Assurance Dashboard will facilitate continuous monitoring, displaying metrics such as model modifications, counterexample handling efficiency, and AGREE usage statistics. This integration aims to quantify the tangible benefits of more explainable counterexamples, driving targeted improvements in usability and overall user experience.

We look forward to exploring these directions in future work and reporting further advancements toward integrating neuro-symbolic verification approaches in MBSE.

Acknowledgments

This work was funded by DARPA contract FA8750-24-9-1000. The views, opinions and/or findings expressed are those of the authors and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government.

References

1. AWS News Blog: Prevent factual errors from llm hallucinations with mathematically sound automated reasoning checks (preview). AWS Blog Post (2024), accessed: 2025-05-11
2. Cofer, D., Gacek, A., Miller, S., Whalen, M.W., LaValley, B., Sha, L.: Compositional verification of architectural models. In: Goodloe, A.E., Person, S. (eds.) NASA Formal Methods. pp. 126–140. Springer (2012)
3. Davis, J.A., Clark, M., Cofer, D., Fifarek, A., Hinchman, J., Hoffman, J., Hulbert, B., Miller, S.P., Wagner, L.: Study on the barriers to the industrial adoption of formal methods. In: Pecheur, C., Dierkes, M. (eds.) Formal Methods for Industrial Critical Systems. pp. 63–77. Springer Berlin Heidelberg (2013)
4. Feiler, P.H., Gluch, D.P.: Model-Based Engineering with AADL: An Introduction to the SAE Architecture Analysis & Design Language. Addison-Wesley Professional, 1st edn. (2012)
5. First, E., Rabe, M.N., Ringer, T., Brun, Y.: Baldur: Whole-proof generation and repair with large language models. arXiv preprint arXiv:2303.04910 (2023)
6. INSPECTA Project: Inspecta. `https://loonwerks.com/projects/inspecta.html`, accessed: 2025-05-19
7. Kaleeswaran, A.P., Nordmann, A., Vogel, T., Grunske, L.: A systematic literature review on counterexample explanation. Information and Software Technology **145** (2022)
8. Megill, N., Wheeler, D.A.: Metamath: A computer language for mathematical proofs (2019)
9. Mirzadeh, S.I., Alizadeh, K., Shahrokhi, H., Tuzel, O., Bengio, S., Farajtabar, M.: GSM-symbolic: Understanding the limitations of mathematical reasoning in large language models. In: The Thirteenth International Conference on Learning Representations (2025), `https://openreview.net/forum?id=AjXkRZIvjB`
10. Object Management Group (OMG): Systems modeling language (sysml) v2 specification. (2024), accessed: 2025-05-19
11. Pei, K., Bieber, D., Shi, K., et al.: Can large language models reason about program invariants? Proceedings of the 40th International Conference on Machine Learning (July 2023)
12. Polu, S., Sutskever, I.: Generative language modeling for automated theorem proving. arXiv preprint arXiv:2009.03393 (2020)
13. RTCA: DO-333: Formal Methods Supplement to DO-178C and DO-278A (December 2011)
14. Sun, C., Sheng, Y., Padon, O., Barrett, C.: Clover: Closed-loop verifiable code generation. arXiv preprint arXiv:2310.17807 (2024)
15. Tahat, A., Hardin, D., Petz, A., Alexander, P.: Metrics for large language model generated proofs in a high-assurance application domain. In: High Confidence Software and Systems Conference (HCSS'24) (2024)
16. Tahat, A., Hardin, D., Petz, A., Alexander, P.: Proof repair utilizing large language models: A case study on the copland remote attestation proofbase. In: Proceedings of International Symposium On Leveraging Applications of Formal Methods Verification and Validation (AISolA) (2024)
17. Wu, H., Barrett, C., Narodytska, N.: Lemur: Integrating large language models in automated program verification. arXiv preprint arXiv:2310.04870 (2023)
18. Zhang, S., First, E., Ringer, T.: Getting more out of large language models for proofs. arXiv preprint arXiv:2305.04369 (2023)

A Appendix

Algorithm 2: Memory Management and Prompt Optimization in AGREE-Dog

Input : User input, conversation state, AADL model repository, optional requirements file

Output: Optimized prompt, updated conversation history

Initialize Short-Term, Temporary, and Long-Term memories;

Identify and load recently updated files:

- Identify recently updated files in repository.
- Load **only** these updated files into Temporary memory.
- Cache filenames and timestamps.

Integrate system-level requirements (if provided);

Construct prompt from:

- Updated files from Temporary memory.
- User input and interaction history.
- System-level requirements.

Ensure prompt size within token limits (truncate oldest entries if necessary);

Generate response from AGREE-Dog model;

Update Short-Term memory with latest interaction;

if ***User selects*** *Save Conversation* **then**

  Save conversation to Long-Term memory;

if ***User selects*** *Commit to Git* **then**

  Stage conversation and updated files;

  Commit and push to remote repository;

return optimized prompt, updated conversation history;

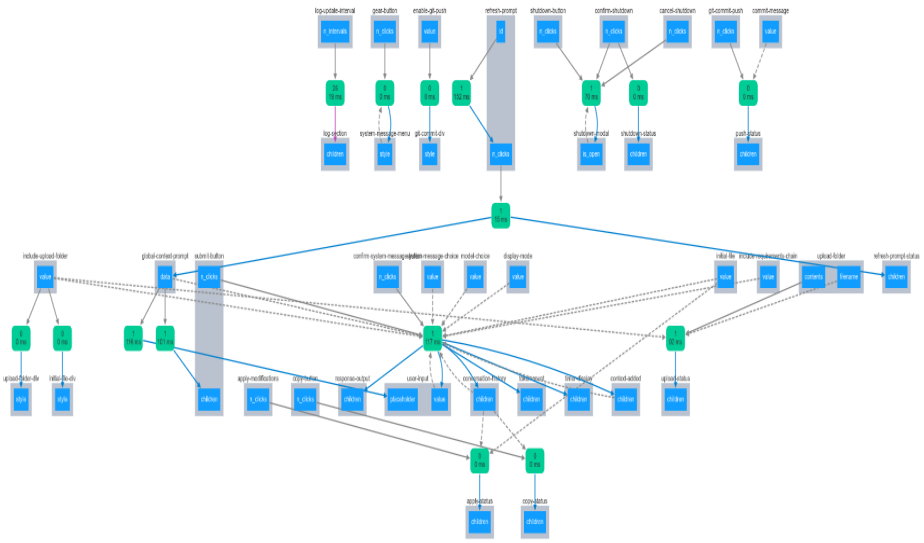


Fig. 7: AGREE-Dog backend function call graph illustrating automated DevOps/ProofOps orchestration. Nodes represent key operations, while edges indicate dependencies and data flows between components.