DejaVuGuard: Runtime Verification of LLM Conversations with First-Order Temporal Logic*

Itay Cohen[1], Klaus Havelund[2], Moran Omer[1], and Doron Peled[1]

[1] Department of Computer Science, Bar-Ilan University, Ramat Gan, Israel
[2] Jet Propulsion Laboratory, California Institute of Technology, Pasadena, USA

Abstract. DejaVuGuard is a runtime verification tool that monitors conversations between a human user and an LLM-based assistant against safety policies written in first-order past-time temporal logic. A separate LLM translates each natural-language message into a structured event describing the relevant facts and objects in that message; an extended version of the DejaVu monitoring engine checks the growing conversation trace incrementally. Policies can therefore refer to concrete objects mentioned in the conversation, such as accounts, patients, ingredients, and monetary amounts. DejaVuGuard can track their identities across messages, and relate them through quantification and numeric comparison. A message that violates an active policy is blocked before it reaches its recipient. The tool ships with a web interface and an end-to-end benchmark of 500 labeled conversations spanning five policy domains; both are openly available.

1 Motivation and Overview

Large language models are increasingly deployed as conversational assistants in customer support, healthcare, finance, and other domains. Many safety requirements in these settings are temporal and object-sensitive: for instance, an assistant may reset an account password only after ownership of that same account was verified. Single-message guardrails cannot express such dependencies or track object identity reliably across turns.

DejaVuGuard applies runtime verification [3] to this problem. It treats a conversation as a finite trace of user and assistant messages and checks it online against policies written in the first-order past-time temporal logic QTL (Quantified Temporal Logic) [6], as supported by the DejaVu monitoring engine [1].

* The research performed by Itay Cohen, Moran Omer and Doron Peled was partially funded by Israeli Science Foundation grant 2454/23: "Validating and controlling software and hardware systems assisted by machine learning". The research performed by Klaus Havelund was carried out at Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not constitute or imply its endorsement by the United States Government or the Jet Propulsion Laboratory, California Institute of Technology.

It extends our earlier propositional conversation monitoring framework [5] to quantified temporal policies over predicates and tracked objects. A dedicated grounding LLM, separate from the assistant under observation, converts each message into the symbolic event representation expected by the monitor. The full approach is described in a companion paper [4]; this abstract presents the tool and its end-to-end benchmark.

The monitor mediates both directions between the user and the assistant: each message is translated into an event, appended to the trace, and checked against the active policies. Since the monitor is external to the assistant LLM, it works with any chat model and can be inspected and modified independently.

2 Specification Language and Monitoring

QTL formulas are interpreted over finite traces of events. In DejaVuGuard, one conversational message contributes one event; an event is a finite set of predicate instances such as $p(a)$ or $q(a, b)$, where p and q are predicate symbols and a, b are concrete values extracted from the conversation. Formulas are built from predicates with variable and constant parameters using Boolean connectives, quantification over the variables, and past-time operators; QTL syntax includes $\boxminus$ (historically), $\diamondminus$ (once), $\mathcal{S}$ (since), and $\ominus$ (previous step). Since the logic is restricted to the past, the monitor has a definitive verdict after each intercepted event. Enforcement can act at the exact message causing a violation.

DejaVu evaluates QTL formulas incrementally over BDD-based representations of value sets, which allows quantification over unbounded domains with finite memory [6]. For DejaVuGuard, the bundled DejaVu engine was extended so that a single event may contain multiple predicate instances, including multiple instances of the same predicate, and it is exposed as an HTTP server managing persistent monitor sessions.

A predicate schema defines a textual criterion for recognizing messages relevant to the policy. It consists of the textual criterion, textual descriptions of the predicate arguments, and a role indicating whether the schema applies to user messages or assistant messages. A policy is a QTL formula whose predicate symbols correspond to the declared predicate schemas.

In order to guard against unwanted behaviors, described using a QTL specification, one needs to translate a conversation with an LLM into a sequence of events. A conversation consists of a sequence of *messages*, where a message may either be the user's prompt or the LLM response. *Grounding* is the message-to-event translation step. For each conversational message, a separate grounding LLM checks the message against each predicate schema used by the active policies. For each schema, it decides whether the message matches the predicate and, if so, extracts the predicate arguments from the text. For each extracted argument, the grounding LLM also produces a normalized value to be used by the monitor; we call this normalization step canonicalization. This normalization is needed because the same object may be described in different ways across the conversation: for example, two messages that mention the same account or patient must contribute the same symbolic value to the trace. After grounding,

the matching predicates and their normalized arguments form the event for the current message. DejaVu consumes this event, updates the monitor state for the active policies, and produces a verdict for each of them. If any active policy receives a negative verdict, DejaVuGuard blocks the conversation.

As an example, consider an account recovery policy:

$$\forall a\, \forall e.\, \Big((\mathsf{add\_email}(a, e) \rightarrow \Diamond\, \mathsf{verified}(a)) \wedge\ (\mathsf{reset}(a) \rightarrow \Diamond\, \mathsf{verified}(a)) \Big).$$

This specification states that adding a recovery email address to an account or resetting its password requires prior ownership verification of that same account.

3 Architecture and Implementation

The implementation is organized around three components.

Backend. The Python backend, built on FastAPI, implements the monitoring pipeline, the grounding module, LLMs configuration, and persistence of predicates, policies, conversations and verdicts in an SQL database.

Frontend. The web-based frontend provides the user-facing control surface and is organized into three main views: Rules, Chat, and Settings. In the Rules view (Figure 1), a supervisor can define predicates, provide their natural-language descriptions and arguments, compose policies over those predicates, validate formulas on the server, and choose which policies are active during monitored chat sessions. In the Chat view, the user tests the active policies by conversing with the assistant through the monitor; each message shows whether it passed or was blocked, and details expose the grounded predicate instances and per-policy verdicts. The Settings view enables selecting the assistant model and the grounding model and exposes the grounding prompt templates for inspection and editing.

Fig. 1. Rules view for defining predicates and active policies.

Monitoring engine. The extended DejaVu engine, implemented in Scala, runs as a separate service and evaluates the QTL specifications. The assistant under observation is accessed through OpenRouter, a hosted gateway for using many

chat-model providers through a common API. The grounding model can run locally through Ollama, LM Studio, or vLLM, which are serving runtimes for local LLM inference. This lets deployments keep monitored conversation content on the local machine when required.

4 End-to-End Monitoring Benchmark

The tool ships with an end-to-end monitoring benchmark of 500 scripted conversations, 3,669 messages in total, organized in five 100-scenario policy families. The families test allergy-aware recipe generation, verified account recovery, medical record access, balance-constrained bank transfers, and budget-constrained car recommendations; together they exercise quantification, the different temporal operators, equality between quantified variables, and numeric comparison over extracted amounts.

Each scenario specifies predicates, a policy, messages, and expected verdicts after each message. The benchmark evaluates the complete pipeline: messages are converted into events, DejaVu checks the growing trace, and every observed verdict is compared with the expected one. The companion paper [4] reports results on a 400-conversation evaluated subset of this benchmark, where DejaVuGuard with the compact LLM `Ministral-3-8B` as grounding model achieves 90.5% scenario accuracy and 98.6% message-level verdict accuracy.

5 Availability

DejaVuGuard is open source and available in the public project repository [2]. The tool resides in the `dejavuguard` directory, and the extended DejaVu engine is bundled with it.

References

1. DejaVu runtime verification tool. `https://github.com/havelund/dejavu`, source code
2. DejaVuGuard: Monitoring LLM conversations with first-order temporal logic. `https://github.com/itay99988/first_order_rv_guardrails` (2026), source code and benchmark data
3. Bartocci, E., Falcone, Y. (eds.): Lectures on Runtime Verification, Introductory and Advanced Topics, Lecture Notes in Computer Science, vol. 10457. Springer (2018)
4. Cohen, I., Havelund, K., Omer, M., Peled, D.: Monitoring LLM conversations with first-order temporal logic (2026), to appear in the proceedings of RV 2026, the 26th International Conference on Runtime Verification
5. Cohen, I., Havelund, K., Omer, M., Peled, D.: Temporal guardrails for LLM conversations: A runtime verification framework. In: AI Verification, Third International Symposium, SAIV 2026. Lecture Notes in Computer Science, vol. 16831, pp. 145–166. Springer (2026)
6. Havelund, K., Peled, D., Ulus, D.: First order temporal logic monitoring with bdds. In: Stewart, D., Weissenbacher, G. (eds.) 2017 Formal Methods in Computer Aided Design, FMCAD 2017, Vienna, Austria, October 2-6, 2017. pp. 116–123. IEEE (2017)