A Certification Scheme for Large Language Models-Based Applications

NICOLA BENA*, Università degli Studi di Milano, Italy
MARCO ANISETTI, Università degli Studi di Milano, Italy and Moon Cloud srl, Italy
ERNESTO DAMIANI†, Università degli Studi di Milano, Italy and Moon Cloud srl, Italy
ALEX DELLA BRUNA, Università degli Studi di Milano, Italy and Consorzio Interuniversitario Nazionale per l'Informatica, Italy
CHAN YEOB YEUN, Khalifa University, UAE
CLAUDIO A. ARDAGNA*, Università degli Studi di Milano, Italy and Moon Cloud srl, Italy

The advent of Large Language Models (LLMs) is revolutionizing the design and deployment of modern applications, enabling intelligent, adaptive, and context-aware services across a wide range of domains. AI-driven services now coexist with legacy systems, microservices, and nanoservices, forming complex and evolving environments. While LLMs offer unprecedented capabilities, they also introduce new risks related to security, privacy, and ethics, especially due to their probabilistic nature and reliance on vast, often opaque, training sets. This paradigm shift calls for robust mechanisms to assess the non-functional behavior of LLM-based applications. In the last decades, assurance has emerged as a key strategy to assess the non-functional properties of complex distributed systems and applications. However, traditional assurance methods prove inadequate to assess LLM-based applications, while existing assurance approaches tailored to LLMs are still in their infancy. In this paper, we propose a multi-dimensional certification scheme for LLM-based applications that initially captures the broader context and dynamic behavior introduced by LLMs. To this aim, after proposing a taxonomy of LLM-based applications and discussing the current gaps in LLM assessment, we develop a certification process that accounts for the peculiarities of the different types of applications in the taxonomy. The certification process builds on a hypergraph that represents a specific behavior supporting the property to be certified and guides the corresponding evidence collection process. We experimentally evaluate our scheme using an LLM-based application that implements an assessment process for cloud systems.

CCS Concepts: • **Software and its engineering → Empirical software validation**; • **Computing methodologies → Artificial intelligence**; **Machine learning**; • **Security and privacy → Software security engineering**.

Additional Key Words and Phrases: Artificial Intelligence, Assurance, Certification, Generative AI, Large Language Models, Security

*Work performed as visiting scholar to C2PS, Computer Science Department, Khalifa University.
†Work performed as Founding Director of C2PS, Computer Science Department, Khalifa University.

Authors' Contact Information: Nicola Bena, nicola.bena@unimi.it, Università degli Studi di Milano, Department of Computer Science, Milan, Italy; Marco Anisetti, marco.anisetti@unimi.it, Università degli Studi di Milano, Department of Computer Science, Milan, Italy and Moon Cloud srl, Milan, Italy; Ernesto Damiani, ernesto.damiani@unimi.it, Università degli Studi di Milano, Department of Computer Science, Milan, Italy and Moon Cloud srl, Milan, Italy; Alex Della Bruna, alex.dellabruna@studenti.unimi.it, Università degli Studi di Milano, Department of Computer Science, Milan, Italy and Consorzio Interuniversitario Nazionale per l'Informatica, Rome, Italy; Chan Yeob Yeun, chan.yeun@ku.ac.ae, Khalifa University, C2PS, Computer Science Department, Abu Dhabi, UAE; Claudio A. Ardagna, claudio.ardagna@unimi.it, Università degli Studi di Milano, Department of Computer Science, Milan, Italy and Moon Cloud srl, Milan, Italy.

Manuscript submitted to ACM

ACM Reference Format:
Nicola Bena, Marco Anisetti, Ernesto Damiani, Alex Della Bruna, Chan Yeob Yeun, and Claudio A. Ardagna. 2026. A Certification Scheme for Large Language Models-Based Applications. *ACM Trans. Intell. Syst. Technol.* 1, 1 (May 2026), 25 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Introduction

Over the past decade, Artificial Intelligence (AI) has profoundly transformed the IT landscape, enabling advanced applications capable of performing tasks that were previously unattainable with traditional control and optimization algorithms. These technologies are now being deployed in critical sectors, such as healthcare [2, 22, 52], military [23], manufacturing [25], transportation [1, 41], and even legal, [12], which were once considered unsuitable for AI bacause of their stringent safety and privacy requirements.

The advent of Large Language Models (LLMs) is further strengthening the IT transformation, enabling intelligent, adaptive, and context-aware applications (LLM-based applications) across a wide range of domains. These applications leverage the capabilities of large language models to understand, generate, or interact with human language in a meaningful way.

While this scenario offers unprecedented opportunities, it also introduces new risks related to security, privacy, and ethics, primarily due to the black-box and probabilistic nature of AI models and LLMs, and their reliance on vast, often opaque, training sets. Concerns about applications driven by non-transparent models have triggered a societal push involving policymakers, regulators, academia, industry stakeholders, and citizens toward the development of trustworthy AI. This movement has culminated in legislative initiatives such as the European Artificial Intelligence Act (Regulation (EU) 2024/1689), which introduces a risk-based framework for AI systems. The regulation defines high-risk applications, mandates transparency for general-purpose AI models, and prohibits certain practices deemed unacceptable, such as emotion recognition in workplaces and biometric categorization based on sensitive attributes [27].

LLMs are reinforcing the need for trustworthy AI with a growing demand for robust assessment of the non-functional behavior of LLM-based applications. In this context, there is an increasing push towards LLM assurance (e.g., certification, compliance), as the way to gain justifiable confidence that LLM-based applications will consistently demonstrate one or more non-functional properties, and operationally behave as expected despite failures and attacks [6]. This also aligns with the European Union AI Act for the responsible development, deployment, and maintenance of AI.

State-of-the-art research on AI sytems assurance is grounded in an assessment process that typically spans the three main phases of the AI life cycle (i.e., data collection and preparation, training, and inference) and establishes a 1:1 mapping between the property to be assessed and the corresponding assessment procedures. Accordingly, applications are verified against a property only if they successfully pass the predefined assessment procedures, regardless of their internal implementation details. Consequently, both the expected application features (as captured by the property definition) and the associated assessment practices must be specified in advance. This implies a fundamentally deterministic view of system behavior, where expected outcomes can be defined a priori and verified through controlled assessment procedures.

However, this deterministic assumption does not align with the intrinsically non-deterministic nature of AI systems, and in particular of LLM-based applications. LLM-based applications introduce new challenges that question several foundational assumptions of existing assurance methodologies. First, they are often opaque, providing limited or no access to information about their pre-training processes, training data, or operational management. Second, they require a reformulation of non-functional properties and the development of novel evidence collection techniques capable of

handling behavioral variability. Finally, they call for an end-to-end assessment approach that spans the entire life cycle of the LLM-based application, including pre-training, adaptation, inference, and the generation of final artifacts.

Although LLM assurance has gained significant attention in recent years, most proposed approaches remain in their infancy. They tend to focus solely on functional properties (e.g., accuracy, performance) [47, 56, 48, 59, 67, 68], evaluate only the final application without considering the full development life cycle [31, 46, 47], and rely on application-specific methodologies [38, 46, 47, 62].

In this paper, we propose a multi-dimensional certification scheme for LLM-based applications that captures the broader context and dynamic behavior introduced by LLMs. Our scheme is built around the notion of application behavior, which binds the property and target of certification to the corresponding evidence collection model. To this end, we first introduce a taxonomy of LLM-based applications, which serves as the foundation for developing a general-purpose certification scheme, and discuss the gaps in existing LLM assessment approaches (Section 2). Next, we define a certification model where the non-functional application behaviors are represented as a hypergraph (Section 3), capturing the specific property to be certified and guiding the evidence collection process that underpins the certification process. We develop the corresponding certification process for LLM-based applications (Section 4) and instantiate it across various application types, covering the full spectrum of LLM-based applications, from agentic AI to classification and generation tasks. Finally, we conduct an extensive experimental evaluation using an LLM-based application that implements a security assessment process for cloud systems.

The contribution of our paper is threefold: *i)* an overview and taxonomy of LLM-based applications and the gaps on LLM assessment, *ii)* the definition of a multi-dimensional certification scheme for LLM-based applications built on a sound definition of certification model, *iii)* the implementation of the certification process built on the scheme at point *ii)* in different scenarios, according to the taxonomy at point *i)*.

2 Background and Motivations

We present a taxonomy of LLM-based applications (Section 2.1) and highlight the gaps in their assessment, which motivate the work presented in this paper (Section 2.2).

2.1 LLM-Based Applications

LLM-based applications leverage the capabilities of large language models to understand, generate, or interact with human language in a meaningful way, enabling features such as text generation, summarization, translation, question answering, artifacts (e.g., code) generation, and more. The life cycle of LLM-based applications is strongly coupled with the life cycle of the corresponding LLMs, which consists of a structured process of three key stages, each serving a distinct purpose, as outlined below.

1. **Pre-training**: the LLM is trained on a large-scale unlabeled dataset, mostly consisting of text. This stage may include additional techniques to improve the LLM performance such as, for instance, *differential privacy*, which introduces noise limiting the information that can be leaked about individuals [35], and *Constitutional AI*, which embeds some human-generated *principles* using a combination of supervised and reinforcement learning [9].
2. **Adaptation**: the LLM is tailored for a specific task and domain [26]. The main adaptation approaches are *fine-tuning* and *prompting*. Fine-tuning adjusts all or a subset of the LLM parameters (e.g., [34, 36]); prompting enriches the input presented to the LLM with *i)* specific training examples, referred to as *in-context learning* [17], *ii)* few learnable parameters to the inputs embeddings [50] or attention layers [44, 42].

3. **Inference**: the LLM receives as input a query and returns as output a response. A query can ask the LLM to perform different activities including classification, summarization, generation of a new artifact. Advanced LLM-based applications apply complex inference pipelines, including: prompt enhancement (e.g., [60]), input extension with additional knowledge (e.g., *Retrieval Augmented Generation* – RAG [43], *Deep Search* [3]), and input and output filtering to block malicious requests (e.g., *Constitutional Classifiers* [9]).[1]

We note that some LLMs do not follow a strict separation between pre-training and adaptation, being trained from scratch with a focus on a specific task and dataset (e.g., *DeepSeek-Coder* [32] and *StarCoder2* [51]) or incorporating large domain-specific training sets (e.g., *BloombergGPT* [65]).

LLM-based applications vary depending on the implementation of the three stages above. With no lack of generality, we identify three representative application types on the basis of the role played by the LLM within the application.

- **Type 1: Standalone**: a standalone foundation LLM. Its life cycle consists of *i)* pre-training on large-scale datasets using a self-supervised learning approach, including the implementation of safety measures, where applicable; *ii)* adaptation using prompting; and *iii)* inference, taking as input human-defined queries and returning as output a digital artifact. The LLM-based application ends with the generation of the digital artifact. Examples include *Claude*, *DeepSeek*, *Gemini*, *GPT-4*, and *Mistral*. We assume user-facing chatbot applications as part of this application type, where the digital artifact produced by an application is directly consumed by the user.
- **Type 2: Integrated – Artifact Generation**: an adapted LLM than generates digital artifacts (e.g., source code, deployment files) according to some specifications. Its life cycle consists of *i)* pre-training of a foundation LLM as in Type 1; *ii)* adaptation using supervised training on a (small) task-specific dataset; and *iii)* inference, taking as input syntactic (i.e., file structure and grammar) and semantic (i.e., purpose) specifications, and returning as output the digital artifact adhering to them. An application Type 2 consists of at least an additional component (e.g., the system orchestrator) that takes as input the digital artifact to implement the application functionalities. Examples include *Code Llama* [58].
- **Type 3: Integrated – Decision-Making**: an LLM driving the application behavior by suggesting the best action to be executed in a given scenario. Its life cycle consists of *i)* pre-training of a foundation LLM as in Type 1; *ii)* adaptation using prompting; and *iii)* inference, taking as input a description of the scenario and possible alternatives, and returning as output the best alternative. The behavior of the application is driven by the decision-making implemented by the corresponding LLM. Examples include LLMs in financial applications [45].

2.2 State of the Art and Gaps on LLM Assessment

The assessment of LLM-based applications (LLM assessment in the following) has received significant attention in recent years (e.g., [31, 46, 47, 57, 67]). One line of research investigates general-purpose LLM assessment techniques and benchmarks. Notably, Liang et al. [47] introduced an assessment process that targets foundation LLMs. The proposed approach builds on a set of use cases linked to specific properties and functions for their assessments, leading to a general, multi-property benchmark useful for downstream applications. White et al. [64] introduced a live benchmark assessing foundation LLMs across different aspects (e.g., reasoning, instruction following). It scores LLMs' outputs against a ground-truth dataset, and ensures that such dataset is not included in any pre-training datasets.

[1]We note that some adaptation techniques such as in-context learning are technically executed at inference time, but are conceptually used for adaptation. For this reason, we consider them part of the latter step.

Another line of research focuses on the usage of LLMs for LLM assessment (*LLM-as-a-Judge*), which, due to their semantics understanding, are particularly suited to assess the free-form textual output of many LLM-based applications. [46]. LLMs can be used to compare outputs (e.g., [10]), and classify the output without reference datasets (e.g., [40, 61]). We refer the reader to the surveys by Gu et al. [31] and Li et al. [46] for more information.

Another line of research investigates the assessment of specific properties of LLM-based applications, notably robustness, fairness, ethics, and privacy. Starting from property robustness, Zhu et al. [70] proposed a benchmark focused on the robustness against prompt manipulations. It includes maligns and benign prompt manipulations across a variety of tasks and settings, including the type of in-context learning. Cantini et al. [18] focused on the robustness against adversarial attacks that aim to elicit existing biases. The proposed approach uses an LLM-as-a-Judge that scores robustness on the basis of the answers to a set of specifically designed prompts. Focusing on property fairness, Bouchard [16] defined a framework that builds on a fine-grained taxonomy of fairness risks, use cases, and metrics verified using specific prompts. The fairness assessment depends on the applications' output only. Binkyte [15], focused on *interactional fairness* in agentic LLM-based applications. Such property models the way agents interact and make cooperative decision, and is assessed using a set of prompts submitted to the agents across different metrics. Fairness assessment in LLM-as-a-Judge has been also widely investigated [62]. Beyond fairness, Jiao et al. [37] investigated ethics, defined as the degree to which the LLM aligns with human ethical standards. The assessment is split into three different dimensions: moral principles, robustness in reasoning, and consistency of values across scenarios. On the other hand, Bahrami et al. [8] viewed, and assessed, ethics under the lens of a number of different issues, including bias, stereotypes, and discrimination. In the context of property privacy, Kim et al. [38] proposed an assessment process that verifies whether an LLM-based application leaks personally identifiable information (PII) that may have been included in the pre-training dataset. The proposed approach submits a set of prompts that include a subset of the PII provided by the data owner/subject, and verifies the presence of additional PII in the retrieved answers. Evertz et al. [28] focused on the confidentiality of the data that are exchanged at inference time in agentic LLM-based applications. The proposed approach embeds a secret key in the application prompt together with instructions not to leak it, and verifies whether the key can still be extracted from the application's output.

Finally, the last line of research focuses on the assessment of LLM-based applications in specific domains. The most notable is software engineering, where LLMs excel at different tasks [33] and there exist several benchmarks and metrics [20].

Despite these substantial advancements in LLM assessment, which provide a solid foundation for LLM assurance, current LLM assurance techniques remain in their infancy. At the same time, traditional assurance methods prove inadequate in this context, leaving a significant *lacuna* in trustworthy AI. We identify five key gaps in LLM assessment that merit particular attention when implementing LLM assurance techniques, as outlined below.

G1: Gap on transparency. Access to the application internals, in terms of pre-training dataset, process, and model structure, varies significantly in LLM assessment. Transparency is entirely missing and details on the pre-training process can only be assumed in closed-source foundation LLMs (e.g., *GPT-4* [55]). On the other hand, few foundation LLMs provide complete access (e.g., *StarCoder2* [51]), while several others sit in the middle ground (e.g., from partial disclosure of pre-training datasets and process only, such as *Claude*, to partial disclosure of pre-training datasets and process with direct model access, such as *Mistral*). This scenario challenges traditional assurance (e.g., [4, 5, 11, 49]), which assumes that the target application can be inspected in its entirety, thus questioning the depth and reliability of current LLM assessment. At the same time, the scale of the pre-training

dataset makes its assessment virtually inapplicable, even when it is fully accessible. Finally, LLMs build on a complex, and (potentially) opaque supply chain, opening up for novel, cascading issues and vulnerabilities [63].

G2: Gap on non-functional assessment. Novel techniques for LLM assessment mostly focus on *functional* assessment, such as the accuracy and performance of the LLM outputs. Traditional assessment, including compliance with policies, risk analysis, and the verification of non-functional attributes such as safety, security, and trustworthiness, tends to be neglected [67] or limited to specific techniques and properties (e.g., [16, 38, 70]). However, they are increasingly mandated by law (e.g., the European Union AI Act) and particularly relevant when LLMs drive the application behavior [47, 67]. In addition, the assessment of accuracy and performance is inapplicable due to the free-form of LLM outputs and complexity of the tasks. The research community is therefore adapting accuracy and performance into several sub-properties such as *coherence*, *consistency*, *fluency*, *relevance* [68], to name but a few.

G3: Gap on unambiguous definition of non-functional properties. Non-functional properties are commonly defined as a property *name* refined with a set of *attributes*, possibly organized in a hierarchy [4, 7]. Such properties are unambiguous and decoupled from the verification process analyzing the deterministic application's outputs. Research on AI model assurance [5, 13] already highlighted that these assumptions do not hold due to the non-deterministic nature of AI. LLMs exacerbate this issue, and introduce the need to verify new properties that *i)* lack a precise definition or common metrics for their assessment, and *ii)* largely depend on the specific domain, task, LLM-based application type, and implementation [47]. For instance, *robustness* is conventionally understood as an application's resilience to malicious activities (e.g., perturbed inputs intended to subvert or evade accurate classification of AI models). Instead, in the context of LLM assessment, robustness can be defined as *i)* the resilience of LLM inference to *benign input perturbations*, such as typos and variations in wording [47], or to *adversarial manipulations*, including *injection* and *jailbreaks* [66], which aim to bypass LLM safeguards and elicit harmful outputs, particularly in scenarios where LLMs directly interact with end users; *ii)* the robustness of the artifacts (e.g., code) generated by the LLM when deployed in the target application. As another example, *bias*/*fairness* are commonly defined as the absence of accuracy disparities across different demographic groups [47]. In the context of LLM assessment, *bias*/*fairness* can be interpreted as the LLM ability to produce outputs that depend solely on the semantic content of the input, rather than being influenced by variations in the prompt structure or wording [62].

G4: Gap on reliable non-functional property assessment. The assessment of LLM-specific non-functional properties typically analyze the *semantics* of the LLM output and requires: *i)* the manual creation of a reference dataset with prompts and expected outputs, and *ii)* the comparison of the retrieved output against the expected one using objective metrics such as, for instance, *BLEU* [56] for translation and *ROUGE* [48] for summarization. This approach suffers from two main drawbacks. First, the dataset must be labeled by humans, which is time-consuming [31]; second, existing metrics perform a *shallow*, syntactic analysis failing to capture semantics [59]. The research community has then started adopting LLM-as-a-Judge [31, 46]. However, research also shows that LLMs are unreliable, because they are strongly influenced by the structure and wording of the target LLMs [46] (e.g., the order of the alternatives [31, 62, 69]). In addition, this paradigm introduces a *chicken-and-egg* problem [46]: an LLM used assessing another LLM must itself be assessed.

G5: Gap on LLM end-to-end assessment. Existing techniques for (non-)functional property assessment (e.g., [16, 47]) collect evidence by analyzing the LLM output only, following G1 and G4. This approach disregards how the LLM has been pre-trained and adapted, which are fundamental steps to assess its non-functional properties.

For instance, the addition of a small fraction of *safe* examples to the adaptation training set can significantly increase *safety* [14]. As another example, the addition of human annotations during training can substantially boost the ethics alignment of the LLM [66]. The way these examples and annotations are generated and added to the training set should therefore be part of the assessment.

Gaps G1–G5 significantly affect the soundness and quality of LLM assessment, thereby diminishing the utility of their results when integrated within LLM assurance. LLM assurance techniques, a pillar of trustworthy AI, cannot assume complete access to the target application (G1) and should focus on non-functional properties, which are increasingly mandated by law and are a major driver of an organization's choices (G2). However, the lack of a clear and commonly agreed-upon set of properties (G3), and corresponding assessment methods (G4), makes it nearly impossible to compare and select LLM-based applications effectively. Moreover, focusing solely on LLM output inspection addresses only the tip of the iceberg in the overall model (and application) life cycle (G5).

In Section 3, we propose a multi-dimensional certification scheme for LLM-based applications that addresses the gaps in this section, supporting the definition of a sound assurance process for LLM-based application.

3 Multi-Dimensional Certification Scheme

A certification scheme implements a certification process that specifies how to verify a given non-functional property p (e.g., fairness) with respect to a target t (e.g., an LLM-based application for secure code generation). To this end, evidence about the behavior of the target is collected according to an evidence collection model $\mathcal{E}$. The property p, the target t, and the evidence collection model $\mathcal{E}$ are defined within a certification model $\mathcal{CM}$ prepared by a certification authority (CA) and executed by an accredited laboratory. If the collected evidence supports property p on target t, the certification authority issues a certificate $\mathcal{C}$.

In this paper, we introduce a multi-dimensional certification scheme based on explicit behavioral modeling to address the gaps on LLM assessment discussed in Section 2.2. The target application is represented as a set of compatible behaviors encoded in a hypergraph, which mitigates the transparency gap (G1) by making verification steps and dependencies explicit, even without full access to the underlying model. Compared to normal graphs, the hypergraphs allow us to model the set of compatible behaviors of the LLM-based application and collect the evidence for its assessment. These behaviors are intrinsically represented as *higher-order relationships*, because *i)* different behaviors jointly occur and are assessed, before moving on to the next behavior and its assessment; and *ii)* alternative behaviors are equally compatible with a given property.

Our certification scheme is explicitly oriented toward non-functional properties, thereby addressing the non-functional assessment gap (G2), and tightly couples non-functional property specifications with their verification logic through a behavior-driven evidence model, overcoming the ambiguities and reliability issues highlighted in G3 and G4. Finally, by spanning pre-training, adaptation, inference, and artifact generation, the scheme enables end-to-end assessment across the entire lifecycle of LLM-based applications, thus addressing G5.

In our multi-dimensional certification, the assessment is executed across four different dimensions as follows.

- **Dimension pre-training d_{pt}:** it verifies the pre-training process, collecting evidence on how pre-training has been executed, and the corresponding training set, if available.
- **Dimension adaptation d_{ad}:** it verifies the adaptation process, collecting evidence on the (supervised) training process, and the corresponding training set.

- **Dimension inference d_{in}:** it verifies the inference process, collecting evidence on the LLM inputs (in terms of examples and prompts) and corresponding outputs.
- **Dimension artifact d_{ar}:** it verifies the artifact returned as output by the LLM as part of the LLM-based application.

Following the existing literature, the certification model in this paper guides the certification activities, although it differs substantially in its structure to correctly represent the target LLM-based applications, whose behavior cannot be fully defined in advance. In fact, the behavior of an LLM-based application: *i)* is not deterministic and *ii)* its expected outputs cannot be defined a priori, *iii)* cannot be modeled through a fixed set of pre-defined attributes, and *iv)* is not unique. Furthermore, LLM assessment cannot be reduced to a simple execution of a set of test cases whose outputs (evidence) are matched against predefined expected results, since LLM outputs may vary across applications and even across executions of the same application. In other words, it is not possible to define a single expected behavior a priori.

Our certification model then defines a set of *compatible application behaviors* that binds the property and the target (i.e., the LLM-based application) to the corresponding evidence collection model in the four dimensions d_{pt}, d_{ad}, d_{in}, and d_{ar}. It is based on the notion of *directed hypergraph* (N, A) where *i)* N=$\{node_1, ..., node_m\}$ is the set of nodes and *ii)* A=$\{arc_1, ..., arc_n\}$ is the set of directed arcs, with each arc_i being a pair of sets $(\{node_i\}, \{node_j\})$ of connected nodes from $\{node_i\}$ (*tail*) to $\{node_j\}$ (*head*) [29]. Notation $tail(arc)$ ($head(arc)$, resp.) denotes the tail (head, resp.) of a hyperarc arc.

Definition 3.1. A certification model $\mathcal{CM}$ is a pair $(p, \mathcal{B})$, where

- p is a textual label describing the non-functional property; and
- $\mathcal{B}$ is a hypergraph (N, A) composed of five sub-hypergraphs $\langle \mathcal{E}_{pt}, \mathcal{E}_{ad}, \mathcal{E}_{in}, \mathcal{E}_{ar}, \mathcal{R} \rangle$ modeling the set of application behaviors compatible with p, where
  - $\mathcal{E}_i = (N_i, A_i)$ is an *evidence collection model hypergraph* for dimension $d_i \in \{d_{pt}, d_{ad}, d_{in}, d_{ar}\}$ in Definition 3.2;
  - $\mathcal{R}$=$(N_{\mathcal{R}}, A_{\mathcal{R}})$ is the *assessment hypergraph* in Definition 3.3.

We note that the property supported by the target LLM-based application is defined by its compatible behaviors. The label p is a short, human-readable description of a property, typically sourced from a shared vocabulary or ontology. It can be modeled in a text-based format such as JSON and used to query specific properties across certificates issued to different applications, thereby supporting the integration of certificates throughout the application lifecycle. We also note that $N = \bigcup_{\forall i} N_i \cup N_{\mathcal{R}}$ and $A = \bigcup_{\forall i} A_i \cup A_{\mathcal{R}}$.

The evidence collection model hypergraph $\mathcal{E}_i$ drives the collection of the evidence in dimension d_i and is defined as follows.

Definition 3.2. An evidence collection model hypergraph $\mathcal{E}_i$ is a hypergraph (N_i, A_i, G) defining how to assess dimension d_i, where

- $N_i = \{v_j\}$ is the set of verification steps, where $v_j = (t, s)$ defines a mechanism $v_j.t$ that is exercised by $v_j.s$ to collect the evidence necessary for the assessment process;
- $A_i = \{arc_j\}$, where $arc_j = (\{v_j\}, \{v_k\})$, is the set of arcs that model the dependencies between the verification steps;
- G is an annotation function that assigns a *guard* $G(v)$ to each verification step v, driving the flow of execution within $\mathcal{E}_i$.

Each $\mathcal{E}_i$ models a set of compatible behaviors in dimension d_i. Each verification step $v_j \in N_i$ specifies the activities $v_j.s$ to be executed on mechanism $v_j.t$ for evidence collection (e.g., testing, monitoring, inspection). We note that $\bigcup_{\forall j} v_j.t$

represents the target of certification (i.e., the LLM-based application). Guard $G(v_j)$ is an expression on evidence collected at step v_j, which determines the flow of execution by selecting the next verification step v_{j+1} to be executed (Section 4).

We recall that the behavior that is exercised during the certification process (see Section 4) is determined at certification time, and cannot be fixed in advance like in traditional certification due to the non-deterministic nature of LLMs. For instance, let us consider an autonomous driving application backed by an LLM [24]. During certification, the verification steps can assess the application's behavior under various conditions, such as the presence of pedestrians, harsh weather, and traffic jams. The observed behavior of the application in these conditions cannot be predetermined and is known only at certification time. Our hypergraph therefore models all compatible behaviors such as, for instance, braking or making a quick swerve.

Assessment hypergraph $\mathcal{R}$ then aggregates the evidence collected according to the hypergraphs $\mathcal{E}_i$ as follows.

Definition 3.3. An assessment hypergraph $\mathcal{R}$ is a hypergraph $(N_{\mathcal{R}}, A_{\mathcal{R}})$ aggregating evidence collected in dimensions $d_i \in \{d_{pt}, d_{ad}, d_{in}, d_{ar}\}$, where:

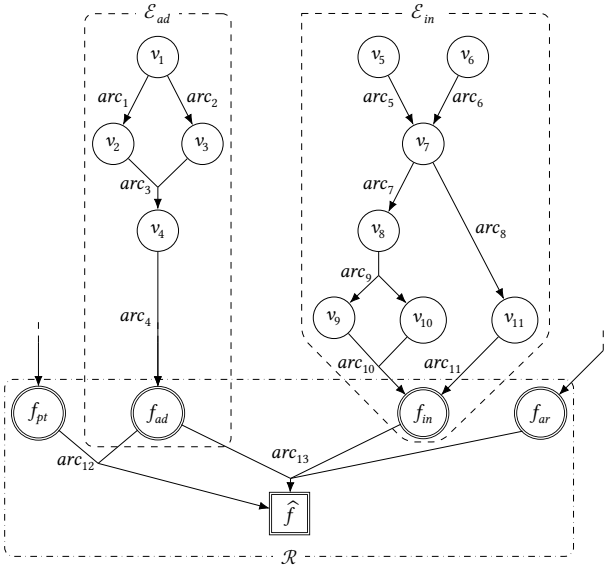
- $N_{\mathcal{R}} = \{f_i\} \cup \{\widehat{f}\}$ is the union of
  - a set $\{f_i\}$ of assessment functions, where each f_i determines the assessment outcome of the corresponding dimension d_i;
  - a global assessment function $\widehat{f}$, which determines the overall assessment outcome;
- $A_{\mathcal{R}} = \left\{(\{v_j\}, \{f_i\})\right\} \cup \left\{(\{f_i\}, \{\widehat{f}\})\right\}$ is the union of
  - hyperarcs linking verification steps $\{v_j\} \subset \mathcal{E}_i.N_i$ without outgoing dependencies to the corresponding assessment function f_i;
  - hyperarcs linking assessment functions f_i to the global assessment function $\widehat{f}$.

We note that the execution flow between the verification steps in $\{\mathcal{E}_{pt}, \mathcal{E}_{ad}\ \mathcal{E}_{in}, \mathcal{E}_{ar}\}$, and the aggregations in $\mathcal{R}$, which jointly model the set of compatible behaviors, are naturally defined as higher-order relations, thereby justifying the usage of a hypergraph. Furthermore, the definition of a property that is strongly coupled with its verification removes all ambiguities addressing G3 and G4, while the explicit description of the application behaviors ensures transparency, addressing G1. Finally, dimensions d_i span the entire lifecycle of LLM-based application and contribute to their non-functional assessment, also addressing G2 and G5.

Example 3.4. Let us consider an LLM-based application of *Type 2: artifact generation* in Section 2.1. The application generates the deployment files of microservices in a distributed system. It is adapted using a dataset of deployment files created by crawling public repositories and, at inference time, takes as input a description of the service to be deployed along with its requirements. The application should be assessed against property *robustness*.

The certification $\mathcal{CM}$ is defined as a pair (*robustness*, $\mathcal{B}$). For simplicity, but no lack of generality, Figure 1 shows an excerpt of $\mathcal{B}$, focusing on dimensions d_{ad} and d_{in}. Evidence collection model $\mathcal{E}_{ad}$ includes v_1, which inspects the sources of the collected dataset. A compatible behavior then includes the usage of *i)* data augmentation (v_2) or *ii)* curriculum learning (v_3) [21].

Evidence collection model $\mathcal{E}_{in}$ specifies verification step v_7, which submits a prompt including a requirement on *high-availability* and verifies that the generated deployment file fulfills it. When the generated file includes multiple replicas without balancing, v_8 verifies whether it includes a way to route traffic between replicas. When the generated file includes an elastic load balancer, v_{11} verifies whether it sets a minimum and maximum number of instances.



Verification steps	
v_1	(sources, repositories-inspection)
v_2	(augmented-dataset, linting)
v_3	(curriculum-learning-proc, incremental-difficulty-inspection)
v_4	(validation-set, diversity-check)
v_5	(modified-prompt, jailbreak-injection)
v_6	(simple-prompt, inference-time)
v_7	(prompt-with-ha-request, ha-inclusion)
v_8	(output-given-prompt, inbound-traffic-inclusion)
v_9	(output-given-prompt, health-check-inclusion)
v_{10}	(output-given-prompt, recovery-inclusion)
v_{11}	(output-given-prompt, load-balancer-threshold-check)

Guards	
$G(v_1)$	• arc_1 if repo includes mostly templates • arc_2 otherwise
$G(v_7)$	• arc_7 if not-balanced replicas included • arc_8 if elastic load balancer included

Fig. 1. Example of $\mathcal{CM}.\mathcal{B}$ (excerpt). Solid-line circles denote verification steps, double-line circles denote assessment functions, double-line square denotes the global assessment function.

Concept	Explanation
Hyperpath *path*	A sequence $(\{node_1\}, arc_{i_1}, \{node_2\}, ..., \{node_{q+1}\})$ of set of nodes and hyperarcs in a hypergraph H s.t. *i)* $\{node_1\} \subseteq tail(arc_{i_1})$, *ii)* $\{node_n\} \subseteq head(arc_{i_q})$, *iii)* $node_k \in head(arc_{i_{k-1}}) \cap tail(arc_{i_k})$ for $k = 2, ..., q$ [29]
Simple hyperpath	A hyperpath where $\forall arc_j, arc_{j+1} \in path, arc_j \neq arc_{j+1}$.
$PathsSimple(H)$	The set $\{path\}$ of simple hyperpaths in hypergraph H
First nodes in *path*	Nodes $\{node_1\}$ in $path=(\{node_1\}, ..., \{node_{q+1}\})$, denoted by $first(path)$.
Last nodes in *path*	Nodes $\{node_{q+1}\}$ in $path=(\{node_1\}, ..., \{node_{q+1}\})$, denoted by $last(path)$.
Source node	A node *node* that does not belong to the head of any hyperarcs, denoted by $source(node)=\top$.

Table 1. Hypergraph notions.

Figure 1 also shows the hypergraph $\mathcal{R}$ in $\mathcal{B}$. It specifies that $\widehat{f}$ can be reached when either dimensions d_{pt} and d_{ad} (arc_{12}) or dimensions d_{ad}, d_{in}, and d_{ar} are verified (arc_{13}).

3.1 Soundness

The soundness of the certification scheme builds on the existence of at least one behavior in $\mathcal{CM}.\mathcal{B}$, called *verification path*, which leads to the certification of the desired property when the corresponding evidence is successfully collected. It starts from a set of verification steps and traverses all their dependencies recursively down to the global assessment function. Starting from the notions in Table 1, a verification path can be defined as follows.

Definition 3.5. Let $\mathcal{CM}$ be a certification model. A verification path $\mathcal{V}$ is a hypergraph $(N_{\mathcal{V}}, A_{\mathcal{V}})$ s.t. the following conditions hold:

1. $N_{\mathcal{V}} \subseteq \mathcal{CM}.\mathcal{B}.N \wedge A_{\mathcal{V}} \subseteq \mathcal{CM}.\mathcal{B}.A$;
2. $\widehat{f} \in N_{\mathcal{V}}$;
3. $\forall v \in N_{\mathcal{V}}, \exists\ path \in PathsSimple(\mathcal{V})$ s.t. $first(path) \in N_{\mathcal{V}} \wedge source(first(path)) = \top \wedge \widehat{f} = last(path)$;

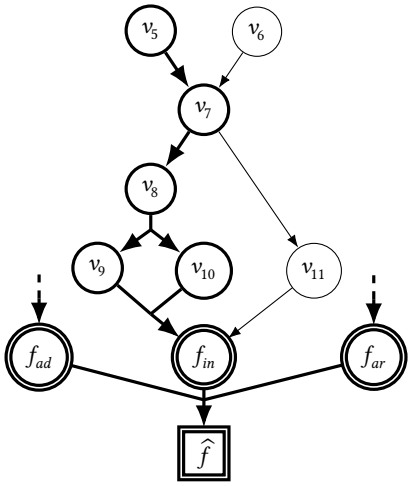


Fig. 2. Example of verification path $\mathcal{V}_1$. Nodes and hyperarcs included in $\mathcal{V}_1$ are represented using thick lines.

4. $N_{\mathcal{V}} = \bigcup_{arc_j \in A_{\mathcal{V}}} tail(arc_j) \cup head(arc_j)$;
5. $\forall node \in N_{\mathcal{V}}$, where *node* is a verification step v, an assessment function f, or a global assessment function $\widehat{f}$, there exists at most one $arc \in A_{\mathcal{V}}$s.t. $node \in tail(arc)$

A verification path $\mathcal{V}$ is a sub-hypergraph of $\mathcal{CM}.\mathcal{B}$ (*Condition 1*) such that *i)* it includes the global assessment function $\widehat{f}$ (*Condition 2*); *ii)* every verification step v is connected to $\widehat{f}$ via a simple hyperpath *path* that starts from a source verification step (*Condition 3*);[2] *iii)* every node *node* belongs to the tail or head of a hyperarc (*Condition 4*); *iv)* every node *node* has only one outgoing arc (*Condition 5*). Condition 5 ensures that in case multiple hyperarcs originate from a verification step, only one is kept in the verification path.

Figure 2 shows an excerpt of $\mathcal{CM}$ in Example 3.4, which includes one verification path $\mathcal{V}_1$ depicted using thick lines. It includes assessment functions f_{ad}, f_{in}, and f_{ar} connected with the global assessment function $\widehat{f}$ using one hyperarc, meaning that the corresponding behavior is compatible with the property when evidence is successfully collected in dimensions d_{ar}, d_{in}, and d_{ar}.

4 Certification Process

The certification process starts when the CA prepares the certification model $\mathcal{CM}$ according to the specifications of the target LLM-based application. The delegated accredited lab then executes the process, which takes as input the certification model, and returns as output a certificate that is signed by the certification authority as follows.

Step (1): Initialization. This step initializes an *annotated verification hypergraph* $\overline{\mathcal{B}}$=($\overline{N}$, $\overline{A}$, G, EV) (verification hypergraph in the following) that is used to collect the evidence supporting the certification process. $\overline{\mathcal{B}}$ extends $\mathcal{CM}.\mathcal{B}$ with an annotation function that assigns the collected evidence $EV(v)$ to the verification step $v \in \overline{N}$. In this step, $\forall v \in \overline{N}, EV(v) = \texttt{nil}$, meaning that each verification step v is initialized with an empty evidence.

[2]We note that, by construction, the source nodes in Definition 3.2 are verification steps.

Step (2): Source selection. This step selects a source verification step v_j such that $EV(v_j) = \texttt{nil}$, meaning that v_j has not been visited yet. If at least one v_j is found the process proceeds to Step (3). Otherwise, if all source verification steps have been visited, it proceeds to Step (5).

Step (3): Depth-first visit. This step collects the evidence according to a depth-first visit of $\overline{\mathcal{B}}$ starting at v_j. Step (3) consists of three separate activities.

i) **Verification step execution** executes $v_j.s$ against mechanism $v_j.t$, collecting an evidence $EV(v_j)$. When the execution of $v_j.s$ fails, $EV(v_j) = \texttt{error}$ and the evidence is tracked separately (e.g., for remediation).

ii) **Evidence analysis** matches collected evidence $EV(v_j)$ against the guard $G(v_j)$, which returns the next hyperarc arc_{j+1} to follow in the verification path. We note that $v_j \in tail(arc_{j+1})$.

iii) **Next hop selection** determines the next node to visit according to the following conditions.

C1) All dependencies met. It means that $head(arc_j)$ and $tail(arc_{j+1})$ have been fully verified. Formally:

$$\forall v_k \in head(arc_j),\ EV(v_k) \notin \{\texttt{error}, \texttt{nil}\} \land \forall v_l \in tail(arc_{j+1}),\ EV(v_l) \notin \{\texttt{error}, \texttt{nil}\}$$

When *C1)* holds and the head of arc_{j+1} includes verification steps only, Step (3) restarts taking as input a verification step $v_l \in head(arc_{j+1})$. Otherwise, when the head of arc_{j+1} corresponds to an assessment function f_i (or a set thereof), a compatible behavior within d_i has been verified, and the process goes to Step (2) to verify other compatible behaviors.

C2) Current dependencies met, forward dependencies unmet. The next hyperarc arc_{j+1} cannot be traversed because its tail has not been fully verified yet. Formally:

$$\forall v_k \in head(arc_j)\, EV(v_k) \notin \{\texttt{error}, \texttt{nil}\} \land \exists\, v_l \in tail(arc_{j+1}),\ EV(v_l) = \texttt{nil}$$

When *C2)* holds, the process goes to Step (2) and visits other verification steps that possibly fully verify the tail of arc_{j+1}.

C3) Current dependencies unmet. Verification steps $\{v_k\}$, reached through hyperarc arc_j, have not been verified yet. Formally:

$$\exists v_k \in head(arc_j),\ EV(v_k) = \texttt{nil}$$

When *C3)* holds, Step (3) restarts taking as input v_k.

C4) Execution failure. The collection of evidence at v_j fails and no hyperarc can be traversed. Formally:

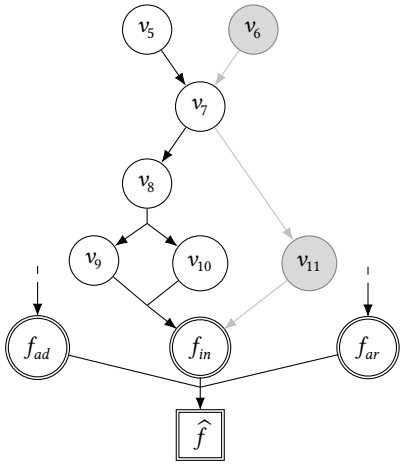
$$EV(v_j) = \texttt{error}$$

When *C4)* holds, the process goes to Step (2) to verify other compatible behaviors.

Jointly, these conditions ensure that the depth-first visit traverses hyperarcs only when the dependencies in their tail are fully verified. We note that when a verification step v_j is visited more than once, Step (3) re-uses the evidence previously collected.

Step (4): Pruning. This step prunes the verification steps in $\overline{\mathcal{B}}$ that did not contribute to the application's assessment. First, Step (4) retrieves the verification steps $\{v_j\}$ s.t. $EV(v_j) = \texttt{nil}$ or $EV(v_j) = \texttt{error}$, and removes them from $\overline{N}$. Then, for each v_j, Step (4) executes three consecutive prunings.

- **Sibling pruning** prunes the verification steps $\{v_k\}$ included in $head(arc_j)$. Specifically, a verification step v_k is pruned *iff* it cannot be reached from anywhere else, that is, *i)* $v_k \in head(arc_j)$; and *ii)* $\forall arc \in \overline{A} \setminus \{arc_j\}$, $v_k \notin head(arc)$.



Collected evidence	
$EV(v_5)$	`jailbreak=fail`
$EV(v_6)$	`error`
$EV(v_7)$	`replicas=11`
$EV(v_8)$	`ingress-controller=nginx`
$EV(v_9)$	`health-check-interval=10s`
$EV(v_{10})$	`restart-policy=always`
$EV(v_{11})$	`nil`

Fig. 3. Example of $\overline{\mathcal{B}}$ (excerpt) following the certification process execution. Pruned nodes and hyperarcs are depicted in grey.

- **Backward pruning** prunes all the hyperarcs terminating at v_j, namely $arc_k \in \overline{A},$ s.t. $v_j \in head(arc_k)$. Then, it examines the nodes in the tails of the pruned hyperarcs; every verification step v_k in the tails is also pruned *iff* it is part of the hyperpath that led to v_j.
- **Forward pruning** prunes all the hyperarcs whose tail includes v_j, namely $arc_{j+1} \in \overline{A},$ s.t. $v_j \in tail(arc_{j+1})$. Then, it examines the nodes in the head of the pruned hyperarcs. Each node of type verification step v_k is also pruned *iff i)* it has not been visited yet, that is, $EV(v_k) = \texttt{nil}$; and *ii)* it does not belong to the head of another hyperarc $arc_k \neq arc_{j+1}$. Each node of type assessment function in $head(arc_{j+1})$ is not pruned.

Sibling, backward, and forward prunings are then recursively re-executed following the dependencies of each pruned v_k. Once the entire $\overline{\mathcal{B}}$ has been traversed, the process goes to Step (5).

Step (5): Aggregation. This step aggregates the evidence collected at Step (3) to verify whether it is sufficient to verify the behavior across all the required dimensions.

Step (5) retrieves the verification paths $\{\mathcal{V}\}$ from $\overline{\mathcal{B}}$. Every path $\mathcal{V}$ represents an application behavior that is finally compatible with the desired property (Definition 3.5). Then, Step (5) eliminates the assessment functions and their associated hyperarcs that do not appear in any verification path, and subsequently transitions to Step (6). Otherwise, when no compatible behavior can be found, the certification process fails.

Step (6): Certificate award. It awards a certificate to the target application. A certificate can be defined as follows.

Definition 4.1. A certificate $\mathcal{C}$ is a pair $(\mathcal{CM}, \overline{\mathcal{B}})$, where $\overline{\mathcal{B}}$ denotes the set of behaviors that are compatible with $\mathcal{CM}$ and have been verified according to the certification process described in this section.

The inclusion of $\mathcal{CM}$ in $\mathcal{C}$ ensures reproducibility of the process, as it permits to compare the set of retrieved compatible behaviors $\mathcal{C}.\overline{\mathcal{B}}$ against the expected behaviors $\mathcal{C}.\mathcal{CM}.\mathcal{B}$. Although different runs of the certification process may produce different compatible behaviors due to non-determinism, $\mathcal{C}.\overline{\mathcal{B}}$ always consists of behaviors included in the space of all compatible behaviors $\mathcal{C}.\mathcal{CM}.\mathcal{B}$. We also note that signatures are affixed to both $\mathcal{CM}$ and $\mathcal{C}$ to ensure their integrity, in line with the literature [7].

Table 2. Mapping assessment dimensions on LLM-based application types.

Type	Dimensions
Type 1: Standalone	• Dimension pre-training d_{pt}: focuses on the pre-training process, if available. • Dimension adaptation and inference d_{ad+in}: focuses on the adaptation and inference processes, which are jointly executed at inference time. • Dimension artifact d_{ar}: focuses on the generated digital artifact consumed by a user.
Type 2: Integrated – Artifact Generation	• Dimension pre-training d_{pt}: focuses on the pre-training process, if available. • Dimension adaptation d_{ad}: focuses on the additional adaptation dataset and how it is used to specialize the LLM. • Dimension inference d_{in}: focuses on the prompt sent to the LLM and corresponding output. • Dimension artifact d_{ar}: focuses on the generated digital artifact and its impact on the application that integrates it.
Type 3: Integrated – Decision-Making	• Dimension pre-training d_{pt}: focuses on the pre-training process, if available. • Dimension adaptation and inference d_{ad+in}: focuses on the adaptation and inference processes, which are jointly executed at inference time. • Dimension artifact d_{ar}: focuses on the generated decision and its impact when applied in the target application.

The certification process is scalable, showing a complexity of $\mathcal{O}(|\overline{N}|^2 \cdot |\overline{A}|)$.[3]

Example 4.2. Figure 3 shows an example of certification process execution on hypergraph $\overline{\mathcal{B}}$. $\overline{\mathcal{B}}$ includes a compatible behavior in the form of verification path $\mathcal{V}_1$ in Figure 2. The verification path includes verification steps v_5, v_7, v_8, v_9, v_{10}, f_{ad}, f_{in}, f_{ar}, $\widehat{f}$, and the corresponding hyperarcs. We note that evidence collection is unsuccessful in v_6 ($EV(v_6)$ =`error`) and v_{11} is not visited ($EV(v_{11})$ =`nil`). As a consequence, v_6, v_{11}, and the corresponding hyperarcs, are pruned (grey nodes and hyperarcs in Figure 3). Evidence collection is successful in the remaining verification steps. The certification process awards a certificate $\mathcal{C}$, which includes the certification model $\mathcal{CM}$ in Example 3.4 and $\overline{\mathcal{B}}$.

To conclude, the compatible behaviors $\mathcal{CM}.\mathcal{B}$ of an LLM-based application and the corresponding certification process executed following $\mathcal{CM}$ vary according to the application type (Section 2.1), as shown in Table 2. When an LLM-based application of Type 1 is assessed, $\mathcal{CM}$ focuses on dimensions pre-training d_{pt}, if available, and an additional dimension d_{ad+in} aggregating dimensions adaptation d_{ad} and inference d_{in}. In fact, Type-1 applications perform adaptation during inference. Finally, $\mathcal{CM}$ focuses also on dimension artifact d_{ar}. When an LLM-based application of Type 2 is assessed, all dimensions, that is, pre-training d_{pt}, adaptation d_{ad}, inference d_{in}, and artifact d_{ar}, are considered. Dimension d_{ar} focuses on the digital artifact generated by the LLM and its impact on the application integrating it. When an LLM-based application of Type 3 is assessed, similarly to Type 1 applications, dimensions adaptation and inference are aggregated (d_{ad+in}). In addition, dimension artifact d_{ar} considers the impact of the generated decision on the application.

These guidelines hold in general; however, when evidence about pre-training and adaptation cannot be collected because the corresponding processes are kept secret, those dimensions are simply excluded from the assessment and, consequently, from $\mathcal{CM}.\mathcal{B}$, or alternatively, assessed solely on the basis of the available documentation. Future developments in mechanistic interpretability could help alleviate this limitation.

5 Experimental Evaluation

We experimentally evaluated our approach by certifying an LLM-based application that implements a security assurance framework for cloud systems. The application builds on agents, called *probes* $\{pr_i\}$, that inspect a given cloud system and evaluate its security posture. It considers a security-critical scenario falling under the high-risk category of the EU AI Act [27].

[3]A detailed complexity analysis is included in our supplementary material.

The application is composed of three distinct modules, which cover the taxonomy of LLM-based applications in Section 2: *i)* a recommendation engine that recommends the set of probes to be executed based on the scenario (Type 3 LLM-based application), *ii)* an explanation engine that interprets the output of the probes and presents it in a user-friendly language (Type 1 LLM-based application), and *iii)* a deployment engine that deploys and executes the selected probes (Type 2 LLM-based application).

We executed our certification process separately on the three modules, using different properties and LLMs (i.e., GPT-OSS, Gemma, Qwen-Coder, CodeLlama, Llama, Deepseek, Mistral). Section 5.1, Section 5.2, and Section 5.3 present the analysis of the certification results for the recommendation engine, the explanation engine, and the deployment engine, respectively. We note that, due to the journal page limit, the detailed discussion of all results retrieved for the recommendation and deployment engines are presented in our supplementary material. In the following, when clear from the context, we use *application* to denote a single module.

Execution environment. Our experiments were executed on a Dell laptop 7590 equipped with a CPU Intel Core i7-9750H with 6 cores@4.5Ghz, 16 GBs RAM, 1 TB M.2 SSD, running RHEL 10.0 and Python v3.12.9. The LLMs were self-hosted and queried via the *Ollama* REST APIs,[4] running on Kubernetes K3S 1.32.4+k3s1. The server is equipped with 2 CPUs AMD EPYC 7453 28-Core (56) @ 3.49 GHz, 256 GBs RAM, 3 GPUs Nvidia L40S, 1 GPU Nvidia A40, and 1 TB M.2 SSD. Datasets, code, and experimental results are available at https://doi.org/10.13130/RD_UNIMI/FCWIXD.

5.1 Type 3: Recommendation Engine

The recommendation engine is a Type 3 LLM-based application that takes as input a description of the scenario and a set of *available probes*, and returns as output a subset of these probes (*selected probes*). Selected probes best match the given scenario, and are chosen to be configured and executed. The application also incorporates *examples* to leverage in-context learning, thereby guiding the LLM toward producing the intended outputs. The certification process focused on property *resilience to order bias*, defined as the extent to which the ordering of the available probes influences the resulting recommendations. *Examples* for in-context learning represent the set $\mathcal{B}$ of behaviors specified in the certification model.

Experimental Protocol. It consists of four steps as follows.

- *Step 1: Probe generation.* This step generates a set of *cloud probes* and a set of *ML probes*; each probe is a pair (*name*, *description*). The former set includes 100 *available probes* applicable to a scenario of security assurance of cloud systems, while the latter set includes 10 probes applicable to a scenario of security assurance of ML systems. The ML probes link the prompt to the specific compatible behavior in $\mathcal{B}$, and are used for defining examples for in-context learning in *Step 2*. The choice of using a distinct set for in-context learning ensures that any bias in the application recommendations is limited to probe ordering and does not stem from additional factors related to the applicability to a specific scenario.
  We generated the two sets using GPT 4.1 and manually revised them to ensure that *i)* each *name* and *description* are consistent, *ii)* each description is error-free, and *iii)* each description only describes well-known technologies relevant to the specific scenario (e.g., object storage, virtual machines, containers), if any. To ensure relevance and consistency, one author (*producer*) manually curated each probes set, while another author (*assessor*) validated the edits. Both authors iteratively reviewed each set until no further modifications were deemed necessary.

[4] https://ollama.com/

Select the m most relevant probes for the scenario CLOUD out of the following n probes:
⟨*Probe list*⟩
Respond on the basis of the following example:
Query: "*Select the m' most relevant probes for the scenario ML out of the following n' probes:*
⟨*Sample probe list*⟩"
Answer: "*The m' most relevant probes to the scenario are:*
⟨*First m' probes*⟩"

Reply only with a JSON object, put the resulting probe names in a field named "`res_probes`". Do not include the description field. Do not add anything other than the JSON object. Ensure that the probe order does not affect your judgment.

Fig. 4. Prompt template with E_{first} example. We formatted the text for clarity.

Select the 3 most relevant probes for the scenario CLOUD out of the following 10 probes:
- `unused_access_keys`: List access keys that have not been used in the last 90 days.
- `unencrypted_storage`: Identify storage volumes not encrypted at rest.
- `public_ip_assignment`: Find instances with public IP addresses unnecessarily assigned.
- [...]

Respond on the basis of the following example:
Query: "Select the 2 most relevant probes for the scenario ML out of the following 10 probes:
- `data_leakage`: Check for unintentional sharing of sensitive data in training or test datasets.
- `model_poisoning`: Test if adversarial samples can corrupt the model during training or inference.
- [...]"

Answer: "The 2 most relevant probes to the scenario are:
- `data_leakage`: Check for unintentional sharing of sensitive data in training or test datasets.
- `model_poisoning`: Test if adversarial samples can corrupt the model during training or inference."

Reply only with a JSON object, put the resulting probe names in a field named "`res_probes`". Do not include the description field. Do not add anything other than the JSON object. Ensure that the order in which the probes are presented does not affect your judgment.

Fig. 5. Excerpt of a prompt example with n=10, m=3, and E_{first}. We formatted the text for clarity.

- *Step 2: Prompt generation.* This step generates the individual prompts that are fed into the application according to the number n of available probes and the number m of selected probes, with $m<n$.
  A prompt consists of n available probes that are randomly selected from the set of cloud probes generated in *Step 1*. It then includes an example of a specific type E that links the prompt to the specific compatible behavior in $\mathcal{B}$. Each example includes *i)* a query asking the most relevant m' probes from a set of n' ML probes generated in *Step 1*, with $m'<n'$; *ii)* a sample answer with the m' selected probes. The example type E determines how the sample answer is defined, as discussed in the experimental settings in Section 5.1.
  Two prompts, $prompt_1(n, m, \{pr\}_2, E)$ and $prompt_2(n, m, \{pr\}_2, E)$, are generated, where $\{pr\}_1$ and $\{pr\}_2$ are created by randomly shuffling the set of n available cloud probes. $\{pr\}_1$ and $\{pr\}_2$ permit to test resilience to ordering bias in application recommendations. Step (2) is executed for each combination of n and m.
  Each generated prompt follows the template shown in Figure 4. Notably, the last line in Figure 4 mitigates the impact of probe ordering on the LLM's output [62].
  Figure 5 presents an excerpt of a *prompt* designed for in-context learning, where the *example* contains a sample answer that selects the first m' probes in the set of n' ML probes.
- *Step 3: Prompt execution.* Each *prompt* is submitted to an application *app* and the corresponding answer is recorded along with the execution time.

Table 3. Experimental settings.

Param.	Values
n	10, 25, 50, 75, 100
m	3, 6, 10, 15
n'	10
m'	2

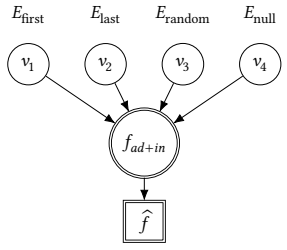
(a)

Example type	Explanation
E_{first}	The chosen probes are the first n' listed among the n' available probes
E_{last}	The chosen probes are the last n' listed among the n' available probes
E_{random}	The chosen probes are n' random listed among the n' available probes
E_{null}	No example

(b)

LLM Name		Scale (N° of param.)	Rel. date
Short	Long		
`gemma3:27b`	`gemma3:27b-it-q4_K_M`	Small–medium (27B)	Feb 2025
`llama3.3:70b`	`llama3.3:70b-instruct-q5_K_M`	Medium–large (70B)	Dec 2024
`deepseek-r1:70b`	`deepseek-r1:70b`	Medium–large (70B)	Jun 2025
`mistral:123b`	`mistral-large:123b-instruct-2411-q5_K_M`	Large (123B)	Nov 2024

(c)



- $\mathcal{V}_1$=($\{v_1\}$, arc_1, $\{f_{ad+in}\}$, arc_5, $\{\widehat{f}\}$)
- $\mathcal{V}_2$=($\{v_2\}$, arc_2, $\{f_{ad+in}\}$, arc_5, $\{\widehat{f}\}$)
- $\mathcal{V}_3$=($\{v_3\}$, arc_3, $\{f_{ad+in}\}$, arc_5, $\{\widehat{f}\}$)
- $\mathcal{V}_4$=($\{v_4\}$, arc_4, $\{f_{ad+in}\}$, arc_5, $\{\widehat{f}\}$)

Fig. 6. $\mathcal{B}$ of the application.

- *Step 4: Assessment.* We denote with $ans(prompt_1, app)$ and $ans(prompt_2, app)$ the answers returned as output by the application app according to $prompt_1(n, m, \{pr\}_1, E)$ and $prompt_2(n, m, \{pr\}_2, E)$, respectively. Each answer ans contains the set of m selected cloud probes recommended by app. Step (4) measures the order bias $bias$ as:

$$bias = 1 - \frac{|ans(prompt_1, app) \cap ans(prompt_2, app)|}{m}, \tag{1}$$

where $bias$=1 means that the order fully affects the answer, $bias$=0 means that it does not.
We note that property resilience to order bias res is calculated by dividing the number of selected probes that appear in both answers by the number m of selected probes, that is, res=$\frac{|ans(prompt_1,app)\cap ans(prompt_2,app)|}{m}$ in equation (1). We note that res=1 models maximum resilience, meaning that the two answers are identical, res=0 models minimum resilience, meaning that the two answers have no probes in common.

Settings. Table 3(a) summarizes the experimental settings. We varied the number n of available cloud probes in $\{10, 15, 50, 75, 100\}$, the number m of selected cloud probes in $\{3, 6, 10, 15\}$. We used n' =10 and m' =2.

Table 3(b) shows the different types of example for in-context learning. Example E_{first} denotes an example designed to induce order bias on the first available ML probes, meaning that the sample answer contains the first m' ML probes in the set of n' available ML probes. Similarly, example E_{last} denotes an example designed to induce order bias on the last available ML probes, meaning that the sample answer contains the last m' ML probes in the set of n' available ML probes. On the other hand, example E_{random} denotes an example designed to not induce order bias, meaning that m' ML probes are randomly extracted from the set of n' available ML probes. Finally, example E_{null} denotes the lack of an example.

The four types of example model four different behaviors of the application in $\mathcal{B}$, which are compatible with property order bias, as depicted in Figure 6. These behaviors model the ability of the application to manage bias in varying contexts. We note that $\mathcal{CM}$ aggregates dimensions adaptation and inference (d_{ad+in}), because the application is of Type 3 (Section 4). Figure 6 also shows the possible verification paths $\mathcal{V}_1$, $\mathcal{V}_2$, $\mathcal{V}_3$, $\mathcal{V}_4$ in $\mathcal{CM}.\mathcal{B}$. Each path evaluates the application order bias with a different example type E, analyzing the set of selected cloud probes varying n and m.

Table 3(c) lists the LLMs considered in our experiments, including one small-to-medium-sized model (`gemma3:27b`), two medium-sized models (`llama3.3:70b` and `deepseek-r1:70b`), and one large-scale model (`mistral:123b`).

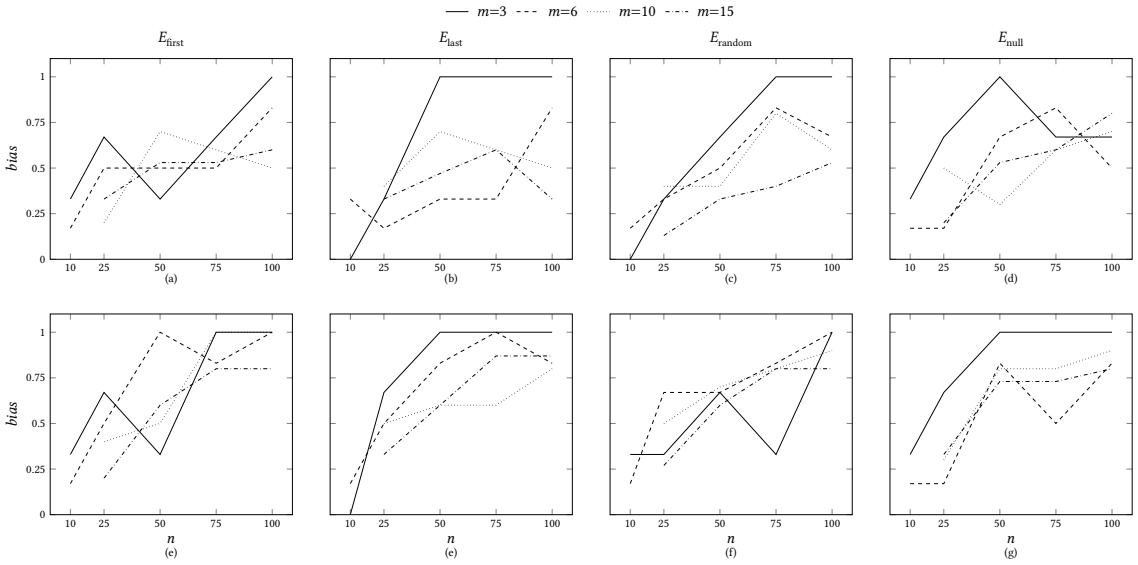


Fig. 7. Order bias measured on `gemma3:27b` (a)–(d) and `llama3.3:70b` (e)–(h) varying n.

We chose this setting to maximize coverage, assess order bias across a wide range of combinations, and introduce reasonable variations in model sizes. This approach advances state-of-the-art assessments of order bias, which predominantly fix n=2 and m=1 [62].

Results. Figures 7–8 show our results varying LLMs; additional results are included in our supplemental material. We note that, when $n \leq m$, no values are plot in Figures 7–8. Figure 7(a)–(d) shows the results of the LLM-based application built on `gemma3:27b`. We can observe that order bias is medium (0.524 on average, with standard deviation σ=0.254). It increases as n increases (0.298 on average with $n \leq 25$ (σ=0.174), compared to 0.637 on average with $n>25$ (σ=0.208)), suggesting that the impact of the probe ordering is higher when many probes are available for selection. We note that the type of example used for in-context learning does not affect the bias. Examples designed to induce order bias (i.e., E_{first} and E_{last}) result in a bias that is similar (0.521, σ=0.251) to that observed with random examples or no examples (i.e., E_{random} and E_{null}) (0.528, σ=0.259). When m increases, the observed bias tends to decrease (Figures 1(e)–(h) in the supplemental material), which may be due to the reduced number of possible recommendations.

Figure 7(e)–(h) shows the results of the LLM-based application built on `llama3.3:70b`. The order bias is generally higher (0.661 on average, (σ=0.279)) compared to `gemma3:27b`, especially as n increases (from 0.362 on average with $n \leq 25$ (σ=0.188) to 0.810 on average with $n>25$ (σ=0.179)). These results suggest that `llama3.3:70b` is more susceptible to order bias with a larger set of available probes. The type of example used for in-context learning affects the bias, with examples designed to induce order bias resulting in higher bias (0.675 on average with E_{first} and E_{last} (σ=0.294) compared to 0.646 on average with E_{random} and E_{null} (σ=0.267)). When m increases, a decreasing trend in bias similar to `gemma3:27b` is observed (Figures 2(e)–(h) in the supplemental material).

Figure 8(a)–(d) shows the results of the LLM-based application built on `deepseek-r1:70b`. The order bias is the highest (0.761 on average (σ=0.219)), with a noticeable increase when $n>25$ (from 0.586 on average with $n \leq 25$ (σ=0.249) to 0.848 on average with $n>25$ (σ=0.137)). The type of example used for in-context learning also significantly influences

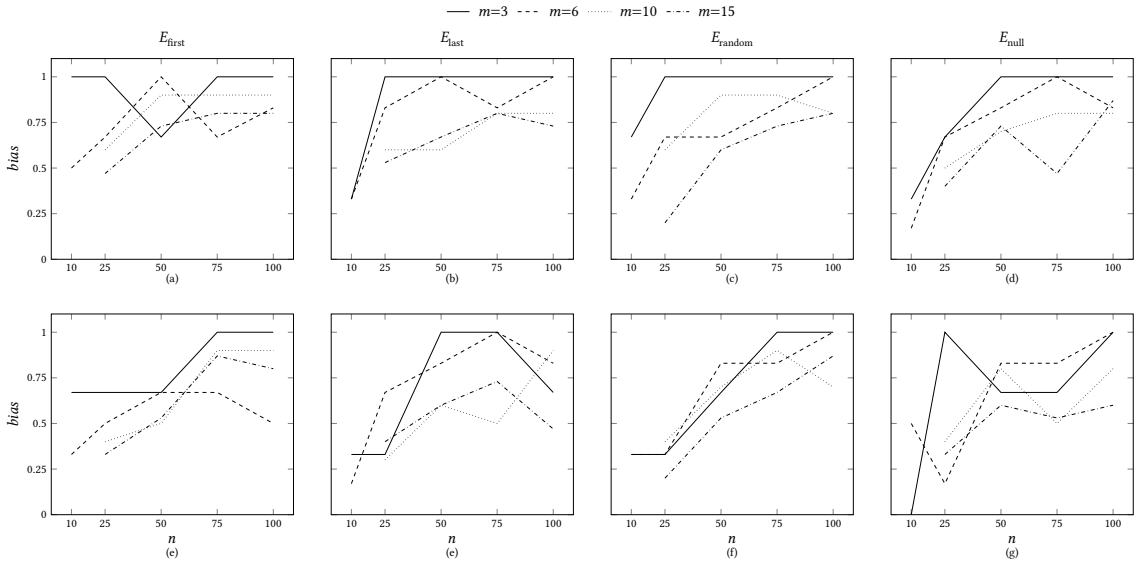


Fig. 8. Order bias measured on `deepseek-r1:70b` (a)–(d) and `mistral:123b` (e)–(h) varying n.

the bias, with examples designed to induce order bias resulting in higher bias (0.786 on average with E_{first} and E_{last} (σ=0.198), compared to 0.735 on average with E_{random} and E_{null} (σ=0.238)). When m increases, the bias decreases, aligning with trends observed in the other applications (Figures 3(e)–(h) in the supplemental material).

Figure 8(e)–(h) shows the results of the LLM-based application built on `mistral:123b`. The order bias is medium (0.640 on average, σ=0.254), with a strong increase as n increases (from 0.393 on average with $n \leq 25$ (σ=0.204) to 0.764 on average with $n>25$ (σ=0.173)). This suggests that `mistral:123b` is affected by the number of available probes. The type of example used for in-context learning marginally affects the bias, with examples designed to induce order bias resulting in slightly higher bias (0.646 on average with E_{first} and E_{last} (σ=0.237), compared to 0.635 on average with E_{random} and E_{null} (σ=0.272)). When m increases, the bias decreases, confirming the trends observed in all applications (Figures 4(e)–(h) in the supplemental material).

We can finally summarize our results as follows. First, all applications exhibit some degree of order bias, which tends to increase as n increases and decrease as m increases. Second, the impact of the type of example used for in-context learning varies according to the LLM: from no to moderate impact when using `gemma3:27b` and `mistral:123b`, to high impact when using `llama3.3:70b` and `deepseek-r1:70b`. Finally, the applications where the impact of the type of example is lower are also those where the order bias is lower. Our results show that an LLM with fewer parameters (i.e., `gemma3:27b`) can be less subject to bias than larger LLMs.

We note that the order bias represents the evidence used for certificate awarding. A threshold of maximum tolerated order bias is defined and a certificate is released *iff* the order bias is less that the maximum threshold.[5] In case the threshold is zero, no requirements are imposed on the measured order bias and a certificate is released any time a verification path in $\overline{\mathcal{B}}$ is verified. For instance, the evidence collected by v_4 in Figure 6 is the average order bias measured

[5]Thresholds are defined using guards $G(v)$ on verification steps (Definition 3.2).

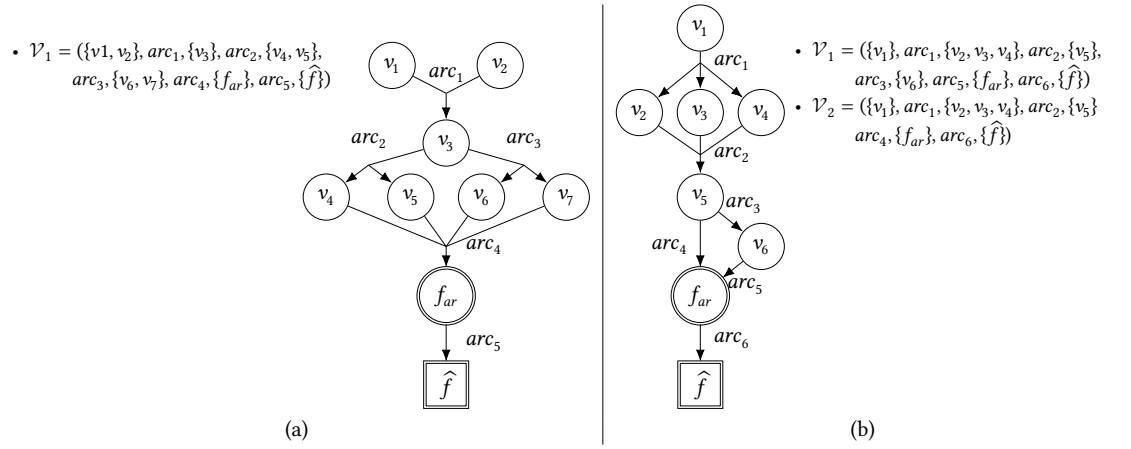


Fig. 9. $\mathcal{B}$ of Type 1 (a) and Type 2 (b) LLM-based applications.

exercising examples of type E_{null}varying n and m. In this case, the application built on `mistral:123b` shows `order-bias`=0.640. The awarded certificate contains the certification model $\mathcal{CM}$ and the annotated verification hypergraph $\overline{\mathcal{B}}$ built during the certification process, according to the collected evidence $EV(v)$.

5.2 Type 1: Explanation Engine

The explanation engine is a Type 1 LLM-based application that takes as input the output of a probe and returns as output a short, user-friendly explanation of the probe output. Explanations are particularly important, because a probe output is a JSON file rich of technicalities that may be difficult to understand for non-expert users. The certification process focused on property *readability*, defined as the degree to which the returned explanation is easy to read and understand for non-expert users.

Experimental protocol. It consists of four steps as follows.

- *Step 1: Probe output collection.* This step collects the probe output. 10 probes $\{pr_1, ..., probe_{10}\}$ are executed against a cloud system; for each probe pr_i, we collected the output of a successful and unsuccessful execution, denoted as $out_i^{\top}$ and $out_i^{\bot}$.
- *Step 2: Prompt generation.* This step generates the set of prompts. For each probe pr_i, we created two prompts: $prompt_1(pr_i, out_i^{\top})$ asking to explain $out_i^{\top}$, and $prompt_2(pr_i, out_i^{\bot})$ asking to explain $out_i^{\bot}$, resulting in a total of 20 prompts.
- *Step 3: Prompt execution.* Each prompt $prompt_i$ is submitted 5 times to the application *app* and the answers recorded (denoted as $\{ans_1(app, prompt_j), ..., ans_5(app, prompt_j)\}$).
- *Step 4: Assessment.* This step assesses the readability of each explanation $ans_k(app, prompt_j)$ using two well-known metrics: *i)* the *Flesch-Kincaid Grade Level* (FKGL), which measures the readability of a text based on the number of syllables per word and words per sentence [39], and *ii)* the *Dale-Chall* readability formula (DC), which measures the readability on the basis of the familiarity of the words against a reference corpus [19]. These metrics measure different aspects of readability, ensuring an objective assessment and limiting potential biases of an LLM judgment [31].

Table 4. Results for the Type 1 (a) and Type 2 (b) applications certification varying the level of detail. In Table 4(a), column "Res." reports whether the assessment was successful (✓) or not (✗), and the normalized number of probes whose explanation was readable; columns "FKGL" and "DC" report the average FKGL and DC scores, resp. (the lower, the better). In Table 4(b), column "Res." reports whether the assessment was successful (✓) or not (✗), and the normalized number of secure deployment files; columns "Mand." and "Opt." report the number of times the three mandatory directives and the optional one were correctly included, resp.

Detail	Res.	FKGL	DC
Low	✗ (0)	9.738	12.157
Medium–Low	✗ (0)	9.671	11.974
Medium-High	✗ (0)	9.895	11.446
High	✗ (0)	11.188	11.608

(a)

Detail	`codellama:70b`			`qwen3-coder:30b`		
	Res.	Mand.	Opt.	Res.	Mand.	Opt.
Low	✗ (0)	0	1	✓(1)	1	1
Medium–Low	✓ (0.4)	0.4	1	✓ (1)	1	1
Medium-High	✓ (0.1)	0.1	1	✓ (1)	1	1
High	✗ (0)	0	1	✓ (1)	1	1

(b)

Settings. Let $FK(ans_k(app, prompt_j))$ and $DL(ans_k(app, prompt_j))$ be the value of FKGL and DL, resp., given the explanation $ans_k(app, prompt_j)$. Figure 9(a) shows the corresponding application behavior $\mathcal{B}$. Verification steps v_1 and v_2 execute the probe and retrieve the output of the successful (v_1) and unsuccessful (v_2) execution, while v_3 generates the corresponding explanation. In verification steps v_4 and v_6, evidence is collected successfully iff $\exists k \in \{1, \dots, 5\}$ s.t. $FK(ans_k(app, prompt_j)) \leq 12$ for all probes and prompts, while in v_5 and v_7, it is collected successfully iff $\exists k \in \{1, \dots, 5\}$ s.t. $DL(ans_k(app, prompt_j)) \leq 8.99$ for all probes and prompts. In other words, the behavior is compatible with property *readability* if the LLM generated one readable explanation of the output of the successful and unsuccessful executions of each probe. The thresholds indicate that an explanation is *readable* if it is understandable by a last-year high school student [19, 39]. We note that this pattern corresponds to the *best-of-n* pattern commonly used when developing LLM-based applications.

Finally, we repeated the above protocol varying the level of prompt detail in {*Low*, *Medium–Low*, *Medium-High*, *High*} using the same set of probes. We configured the application *app* using the LLM `gpt-oss:20b` because it does not include safeguards, which can easily conflict with the successful completion of the required task. Figure 6 in the supplemental material shows the prompts templates.

Results. Table 4(a) summarizes our results. Column "Res." reports whether the assessment was successful (✓) or not (✗), and the (normalized) number of probes whose explanation was readable (i.e, $FK(ans_k(app, prompt_j)) \leq 12$ and $DL(ans_k(app, prompt_j)) \leq 8.99$); columns "FKGL" and "DC" report the average FKGL and DC scores, respectively. We can first observe that the application does not support the required property, because the value of DC is always higher than the threshold; this means that, while the explanation is syntactically readable (low FKGL, <12), it consists of many unfamiliar words (high DC, >11). In addition, we can observe that when the level of detail of the prompt is *High*, the readability further worsens. These results are in line with the literature, where some details in the prompt increase accuracy but only up to a certain point [71], though this phenomenon has been rarely studied in the context of non-functional properties (e.g., [53]).

5.3 Type 2: Deployment Engine

The deployment engine is a Type 2 LLM-based application that takes as input a probe and returns as output the corresponding deployment file (a Kubernetes YAML file) for its execution on the application infrastructure. The certification process focused on property *security*, defined as the degree to which the returned deployment file is secure, in terms of adoption of hardening practices (e.g., running the probe with non-root privileges).

Experimental protocol. It consists of three steps as follows.

- *Step 1: Prompt generation.* This step defines a prompt *prompt* asking to generate a Kubernetes deployment file for a given probe *pr*; we did not vary *pr* because the prompt content is independent of the specific probe.
- *Step 2: Prompt execution.* Each *prompt* is submitted 10 times and the answers recorded. This repetition was necessary because LLMs struggle in generating a valid YAML.
- *Step 3: Assessment.* This step assesses the security of the deployment following the behavior $\mathcal{B}$ in Figure 9(b), which verifies *i)* the mandatory presence of directives limiting the container *capabilities* (v_2), file system access (v_3), and running user (v_4); and *ii)* the optional presence of a sandboxing directive (v_5 and v_6).

Settings. Following Section 5.2, we repeated the protocol varying the level of prompt detail in {*Low*, *Medium–Low*, *Medium-High*, *High*}. We configured the application *app* using LLMs specialized in code generation, namely `codellama:70b` and `qwen3-coder:30b`. According to Figure 9(b), there are two behaviors compatible with property *security*. The first one consists of verification path $\mathcal{V}_1$=($\{v_1\}$, arc_1, $\{v_2, v_3, v_4\}$, arc_2, $\{v_5\}$, arc_3, $\{v_6\}$, arc_5, $\{f_{ar}\}$, arc_6, $\{\widehat{f}\}$), and models the simultaneous presence of the three mandatory directives and the sandboxing directive (which, when present, must also be correctly configured) in at least one generated file. The second one consists of $\mathcal{V}_2$=($\{v_1\}$, arc_1, $\{v_2, v_3, v_4\}$, arc_2, $\{v_5\}$, arc_4, $\{f_{ar}\}$, arc_6, $\{\widehat{f}\}$) and models the presence of the three mandatory directives only in at least one generated file. Figure 7 in the supplemental material shows the prompts templates, while Table 3 in the supplemental material shows the expected directives in detail.

Results. Table 4(b) summarizes our results. Column "Res." reports the (normalized) number of times the generated deployment was secure (i.e., the deployment limits capabilities, file system access, and running user, and, optionally, utilizes sandboxing); columns "Mand." and "Opt." report the normalized number of times the three mandatory directives and the optional one were correctly used. We can first observe that results substantially vary between `codellama:70b` and `qwen3-coder:30b`, with the assessment succeeding in half of the cases when *app* is configured using `codellama:70b`, always succeeding when configured using `qwen3-coder:30b`. This result is expected, because `qwen3-coder:30b` was released 2 years after `codellama:70b`. A closer look at the results in Table 4 in the supplemental material reveals that `codellama:70b` struggles even with the mandatory directives, with the configuration of the non-root user being the most frequently used (35% on average compared to 25% of read-only file system and 15% of capabilities limiting), while `qwen3-coder:30b` always includes the three directives. Interestingly, `codellama:70b` never includes the sandboxing directive, while `qwen3-coder:30b` always includes the sandboxing directive, except when the level of detail is *Low*.

6 Discussion

LLMs are becoming the cornerstone of modern AI-based applications, enabling forms of task execution and knowledge integration that were unimaginable until few years ago. Despite already exhibiting human-like quality, their non-functional assessment, which is increasingly mandated by law and regulations, remains more an art than a science. The certification scheme in this paper departs from state-of-the-art assurance techniques by addressing the inherent non-deterministic behavior of LLM-based applications. Our scheme is built around the notion of compatible application behavior, which binds the required property and target of certification to the corresponding evidence collection model. The certification scheme implements a certification process that verifies the precise application behavior across multiple dimensions that impact the assessment.

Our scheme is the first approach to the non-functional certification of LLM-based applications, which initially addresses the gaps on LLM assessment in Section 2.2 as follows. First, the explicit modeling of the set of compatible

behaviors provides a clear representation of the application and its life cycle. This representation, along with the specific behaviors, is included in the awarded certificates, and can therefore be inspected and used for certification-based decision-making (G1 and G5). Second, the definition of the property *in terms* of the application behavior and its assessment removes the assumption of properties whose definition is applicable to any scenarios (G2, G3 and G4).

Nevertheless, the present scheme introduces some tradeoffs in addressing the gaps, which will be the target of our future work. First, transparency of LLMs can still represent an obstacle to LLM-based application certification. Our future work will propose a certification process for LLM-based application that supports partial assessment when dimensions pre-training and adaption are inaccessible. We will also investigate the integration of *data* [30] and *model cards* [54], as well as evidence collection based on mechanistic interpretability. Second, the applicability of our approach in the real world is still limited. Our future work will investigate *templates* of compatible behaviors that can then be instantiated in different scenarios, to support a simplified certification process. We will also investigate how to increase the automation of the certification process, which currently places a significant burden on the CA due to the need to manually define the complete set of compatible behaviors. In this context, we will investigate semi-automatic generation techniques following a human-in-the-loop approach, which is fundamental to guarantee a level of correctness and trustworthiness. Finally, we will study the chain of trust of our certification scheme, where the role of each participating actor and the rules governing the certification life cycle are thoroughly articulated.

Acknowledgments

Research supported, in parts, by *i)* TII under Grant 8434000394; *ii)* project BA-PHERD, funded by the European Union – NextGenerationEU, under the National Recovery and Resilience Plan (NRRP) Mission 4 Component 2 Investment Line 1.1: "Fondo Bando PRIN 2022" (CUP G53D23002910006); *iii)* Piano di sostegno alla ricerca, Università degli Studi di Milano; *iv)* PSR 2025 – Linea 8 – Sottomisura A, Università degli Studi di Milano. Computational resources provided by INDACO Core facility, which is a project of High Performance Computing at the University of MILAN http://www.unimi.it.

References

[1] Arulmurgan Aalavanthar, S Famila, Shanmugam Sundaramurthy, Stefano Cirillo, Giandomenico Solimando, and Giuseppe Polese. 2025. Multi-objective federated learning traffic prediction in vehicular network for intelligent transportation system. *PeerJ Computer Science*, 11.

[2] Antonio Agliata, Antonio Pilato, Sorrentino Mariacarmen, Salvatore Bottiglieri, Emanuel Di Nardo, and Angelo Ciaramella. 2024. Generative ai and emotional health: innovations with haystack. In *Proc. of IEEE ISCC 2024*. Paris, France, (June 2024).

[3] Salaheddin Alzubi et al. 2025. Open deep search: democratizing search with open-source reasoning agents. *arXiv preprint arXiv:2503.20201*.

[4] Marco Anisetti, Claudio A. Ardagna, and Nicola Bena. 2023. Multi-dimensional certification of modern distributed systems. *IEEE Transactions on Services Computing*, 16, 3.

[5] Marco Anisetti, Claudio A. Ardagna, Nicola Bena, and Ernesto Damiani. 2023. Rethinking certification for trustworthy machine-learning-based applications. *IEEE Internet Computing*, 27, 6.

[6] Claudio A. Ardagna, Rasool Asal, Ernesto Damiani, and Quang Hieu Vu. 2015. From Security to Assurance in the Cloud: A Survey. *ACM CSUR*, 48, 1.

[7] Claudio A. Ardagna and Nicola Bena. 2023. Non-functional certification of modern distributed systems: a research manifesto. In *Proc. of IEEE SSE 2023*. Chicago, IL, USA, (July 2023).

[8] Mehdi Bahrami, Ryosuke Sonoda, and Ramya Srinivasan. 2024. Llm diagnostic toolkit: evaluating llms for ethical issues. In *Proc. of IJCNN 2024*. Yokohama, Japan, (June 2024).

[9] Yuntao Bai et al. 2022. Constitutional ai: harmlessness from ai feedback. *arXiv preprint arXiv:2212.08073*.

[10] Yushi Bai et al. 2023. Benchmarking foundation models with language-model-as-an-examiner. In *Proc. of NeurIPS 2023*. New Orleans, LA, USA, (Dec. 2023).

[11] Christian Banse, Björn Fanta, Juncal Alonso, and Cristina Martinez. 2025. EMERALD: Evidence Management for Continuous Certification as a Service in the Cloud. In *Proc. of CLOSER 2025*. Porto, Portugal, (Apr. 2025).

[12] Valerio Bellandi, Samira Maghool, and Stefano Siccardi. 2023. An nlp-based statistical reporting methodology applied to court decisions. In *Proc. of Euromicro SEAA 2023*. Durres, Albania, (Sept. 2023).

[13] Nicola Bena, Marco Anisetti, Gabriele Gianini, and Claudio A. Ardagna. 2024. Certifying accuracy, privacy, and robustness of ml-based malware detection. *SN Computer Science*, 5.

[14] Federico Bianchi, Mirac Suzgun, Giuseppe Attanasio, Paul Röttger, Dan Jurafsky, Tatsunori Hashimoto, and James Zou. 2024. Safety-tuned llamas: lessons from improving the safety of large language models that follow instructions. *arXiv preprint arXiv:2309.07875*.

[15] Ruta Binkyte. 2025. Interactional fairness in llm multi-agent systems: an evaluation framework. *arXiv preprint:2505.12001v1*.

[16] Dylan Bouchard. 2024. An actionable framework for assessing bias and fairness in large language model use cases. *arXiv preprint:2407.10853*.

[17] Tom Brown et al. 2020. Language models are few-shot learners. In *Proc. of NeurIPS 2020*. Virtual, (Dec. 2020).

[18] Riccardo Cantini, Alessio Orsino, Massimo Ruggiero, and Domenico Talia. 2025. Benchmarking adversarial robustness to bias elicitation in large language models: scalable automated assessment with llm-as-a-judge. *arXiv preprint:2504.07887*.

[19] Jeanne Sternlicht Chall and Edgar Dale. 1995. *Readability Revisited: The New Dale-Chall Readability Formula*. Brookline Books, 159.

[20] Liguo Chen et al. 2025. A survey on evaluating large language models in code generation tasks. *arXiv preprint arXiv:2408.16498*.

[21] Xiaoyin Chen, Jiarui Lu, Minsu Kim, Dinghuai Zhang, Jian Tang, Alexandre Piché, Nicolas Gontier, Yoshua Bengio, and Ehsan Kamalloo. 2025. Self-evolving curriculum for llm reasoning. *arXiv preprint arXiv:2505.14970*.

[22] S. Cirillo, S. C. Rajkumar, L. Solimando, and D. Yuvasini. 2025. A hybrid approach combining images and questionnaires for early detection and severity assessment of Autism Spectrum Disorder. *Image and Vision Computing*, 160.

[23] L. Colombi, S. Dahdal, E. Di Caro, R. Fronteddu, and A. Gilli. 2024. Efficient Data Dissemination via Semantic Filtering at the Tactical Edge. In *Proc. of IEEE MILCOM 2024*. Washington, DC, USA, (Oct.–Nov. 2024).

[24] Can Cui et al. 2024. A survey on multimodal large language models for autonomous driving. In *Proc. of IEEE/CVF WACVW 2024*. Waikoloa, HI, USA, (Jan. 2024).

[25] Simon Dahdal, Lorenzo Colombi, Matteo Brina, Alessandro Gilli, Mauro Tortonesi, Massimiliano Vignoli, and Cesare Stefanelli. 2024. An mlops framework for gan-based fault detection in bonfiglioli's evo plant. *Infocommunications Journal*, 16, 2.

[26] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: pre-training of deep bidirectional transformers for language understanding. In *Proc. of NAACL 2019*. Minneapolis, Minnesota, (June 2019).

[27] European Parliament and Council. 2024. Regulation (eu) 2024/1689 of the european parliament and of the council of 13 june 2024 laying down harmonised rules on artificial intelligence and amending regulations (ec) no 300/2008, (eu) no 167/2013, (eu) no 168/2013, (eu) 2018/858, (eu) 2018/1139 and (eu) 2019/2144 and directives 2014/90/eu, (eu) 2016/797 and (eu) 2020/1828 (artificial intelligence act). (2024). https://eur-lex.europa.eu/eli/reg/2024/1689/oj/eng.

[28] Jonathan Evertz, Merlin Chlosta, Lea Schönherr, and Thorsten Eisenhofer. 2025. Whispers in the machine: confidentiality in agentic systems. *arXiv preprint arXiv:2402.06922*.

[29] Giorgio Gallo, Giustino Longo, Stefano Pallottino, and Sang Nguyen. 1993. Directed hypergraphs and applications. *Discrete Applied Mathematics*, 42, 2–3.

[30] Timnit Gebru, Jamie Morgenstern, Briana Vecchione, Jennifer Wortman Vaughan, Hanna Wallach, Hal Daumé III, and Kate Crawford. 2021. Datasheets for datasets. *CACM*, 64, 12.

[31] Jiawei Gu et al. 2025. A survey on llm-as-a-judge. *arXiv preprint arXiv:2411.15594*.

[32] Daya Guo et al. 2024. Deepseek-coder: when the large language model meets programming – the rise of code intelligence. *arXiv preprint arXiv:2401.14196*.

[33] Junda He, Christoph Treude, and David Lo. 2025. Llm-based multi-agent systems for software engineering: literature review, vision, and the road ahead. *ACM TOSEM*, 34, 5.

[34] Dan Hendrycks, Xiaoyuan Liu, Eric Wallace, Adam Dziedzic, Rishabh Krishnan, and Dawn Song. 2020. Pretrained transformers improve out-of-distribution robustness. In *Proc. of ACL 2020*. Online, (July 2020).

[35] Shlomo Hoory et al. 2021. Learning and evaluating a differentially private pre-trained language model. In *Proc. of EMNLP 2021*. Punta Cana, Dominican Republic, (Nov. 2021).

[36] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. Lora: low-rank adaptation of large language models. In *Proc. of ICLR 2022*. Virtual, (Apr. 2022).

[37] Junfeng Jiao, Saleh Afroogh, Abhejay Murali, Kevin Chen, David Atkinson, and Amit Dhurandhar. 2025. Llm ethics benchmark: a three-dimensional assessment system for evaluating moral reasoning in large language models. *Scientific Reports*, 15, 1.

[38] Siwon Kim, Sangdoo Yun, Hwaran Lee, Martin Gubri, Sungroh Yoon, and Seong Joon Oh. 2023. Propile: probing privacy leakage in large language models. In *Advances in Neural Information Processing Systems*. Curran Associates, Inc.

[39] J. Peter Kincaid, Jr. Fishburne Robert P., Richard L. Rogers, and Brad S. Chissom. 1975. Derivation of New Readability Formulas (Automated Readability Index, Fog Count and Flesch Reading Ease Formula) for Navy Enlisted Personnel. Research Branch Report ADA006655. Naval Technical Training Command, Research Branch, (Jan. 1975). https://apps.dtic.mil/sti/pdfs/ADA006655.pdf.

[40] Tom Kocmi and Christian Federmann. 2023. Large language models are state-of-the-art evaluators of translation quality. In *Proc. of EAMT 2023*. Tampere, Finland, (June 2023).

[41] Alessandro La Ferlita, Yan Qi, Emanuel Di Nardo, Ould El Moctar, Thomas E. Schellin, and Angelo Ciaramella. 2024. A framework of a data-driven model for ship performance. *Ocean Engineering*, 309.

[42] Brian Lester, Rami Al-Rfou, and Noah Constant. 2021. The power of scale for parameter-efficient prompt tuning. In *Proc. of EMNLP 2021*. Punta Cana, Dominican Republic, (Nov. 2021).

[43] Patrick Lewis et al. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. In *Proc. of NeurIPS 2020*. Virtual, (Dec. 2020).

[44] Xiang Lisa Li and Percy Liang. 2021. Prefix-tuning: optimizing continuous prompts for generation. In *Proc. of IJCNLP 2021*. Virtual, (Aug. 2021).

[45] Yinheng Li, Shaofei Wang, Han Ding, and Hang Chen. 2023. Large language models in finance: a survey. In *Proc. of ACM ICAIf 2023*. Brooklyn, NY, USA, (Nov. 2023).

[46] Zhen Li, Xiaohan Xu, Tao Shen, Can Xu, Jia-Chen Gu, Yuxuan Lai, Chongyang Tao, and Shuai Ma. 2024. Leveraging large language models for NLG evaluation: advances and challenges. In *Proc. of EMNLP 2024*. Miami, FL, USA, (Nov. 2024).

[47] Percy Liang et al. 2023. Holistic evaluation of language models. *TMLR*.

[48] Chin-Yew Lin. 2004. ROUGE: A package for automatic evaluation of summaries. In *Proc. of Workshop on Text Summarization Branches Out*. Barcelona, Spain, (July 2004).

[49] Sebastian Lins, Stephan Schneider, and Ali Sunyaev. 2018. Trust is Good, Control is Better: Creating Secure Clouds by Continuous Auditing. *IEEE Transactions on Cloud Computing*, 6, 3.

[50] Xiao Liu, Kaixuan Ji, Yicheng Fu, Weng Tam, Zhengxiao Du, Zhilin Yang, and Jie Tang. 2022. P-tuning: prompt tuning can be comparable to fine-tuning across scales and tasks. In *Proc. of ACL 2022*. Dublin, Ireland, (May 2022).

[51] Anton Lozhkov et al. 2024. Starcoder 2 and the stack v2: the next generation. *arXiv preprint arXiv:2402.19173*.

[52] Chee-Peng Lim, Ashlesha Vaidya, Nikhil Jain, Margarita N. Favorskaya, and Lakhmi C. Jain, (Eds.) 2024. *Technologies and strategies for continuous learning through electronic health records data. Advances in Intelligent Healthcare Delivery and Management: Research Papers in Honour of Professor Maria Virvou for Invaluable Contributions*. Springer Nature Switzerland, Cham, 1–36. ISBN: 978-3-031-65430-5. doi:10.1007/978-3-031-65430-5_1.

[53] Fiammetta Marulli, Lelio Campanile, Maria Stella de Biase, Stefano Marrone, Laura Verde, and Marianna Bifulco. 2024. Understanding readability of large language models output: an empirical analysis. *Procedia Computer Science*, 246.

[54] Margaret Mitchell, Simone Wu, Andrew Zaldivar, Parker Barnes, Lucy Vasserman, Ben Hutchinson, Elena Spitzer, Inioluwa Deborah Raji, and Timnit Gebru. 2019. Model cards for model reporting. In *Proc. of FAT* 2019*. Atlanta, GA, USA, (Jan. 2019).

[55] OpenAI et al. 2024. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

[56] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proc. of ACL 2002*. Philadelphia, PA, USA, (July 2002).

[57] Ji-Lun Peng, Sijia Cheng, Egil Diau, Yung-Yu Shih, Po-Heng Chen, Yen-Ting Lin, and Yun-Nung Chen. 2024. A survey of useful llm evaluation. *arXiv preprint:2406.00936*.

[58] Baptiste Rozière et al. 2024. Code llama: open foundation models for code. *arXiv preprint arXiv:2308.12950*.

[59] Natalie Schluter. 2017. The limits of automatic summarisation according to ROUGE. In *Proc. of EACL 2017: Volume 2, Short Papers*. Valencia, Spain, (Apr. 2017).

[60] Tobias Schnabel and Jennifer Neville. 2024. Symbolic prompt program search: a structure-aware approach to efficient compile-time prompt optimization. *arXiv preprint arXiv:2404.02319*.

[61] Jiaan Wang, Yunlong Liang, Fandong Meng, Zengkui Sun, Haoxiang Shi, Zhixu Li, Jinan Xu, Jianfeng Qu, and Jie Zhou. 2023. Is ChatGPT a good NLG evaluator? a preliminary study. In *Proc. of NewSum 2023*. Singapore, (Dec. 2023).

[62] Peiyi Wang et al. 2024. Large language models are not fair evaluators. In *Proc. of ACL 2024*. Bangkok, Thailand, (Aug. 2024).

[63] Shenao Wang, Yanjie Zhao, Xinyi Hou, and Haoyu Wang. 2025. Large language model supply chain: a research agenda. *ACM TOSEM*, 34, 5.

[64] Colin White et al. 2025. Livebench: a challenging, contamination-limited llm benchmark. In *Proc. of ICLR 2025*. Singapore, (Apr. 2025).

[65] Shijie Wu, Ozan Irsoy, Steven Lu, Vadim Dabravolski, Mark Dredze, Sebastian Gehrmann, Prabhanjan Kambadur, David Rosenberg, and Gideon Mann. 2023. Bloomberggpt: a large language model for finance. *arXiv preprint arXiv:2303.17564*.

[66] Yifan Yao, Jinhao Duan, Kaidi Xu, Yuanfang Cai, Zhibo Sun, and Yue Zhang. 2024. A survey on large language model (llm) security and privacy: the good, the bad, and the ugly. *High-Confidence Computing*, 4, 2.

[67] Asaf Yehudai, Lilach Eden, Alan Li, Guy Uziel, Yilun Zhao, Roy Bar-Haim, Arman Cohan, and Michal Shmueli-Scheuer. 2025. Survey on evaluation of llm-based agents. *arXiv preprint arXiv:2503.16416*.

[68] Weizhe Yuan, Graham Neubig, and Pengfei Liu. 2021. Bartscore: evaluating generated text as text generation. In *Proc. of NeurIPS 2021*. Virtual, (Dec. 2021).

[69] Lianmin Zheng et al. 2023. Judging llm-as-a-judge with mt-bench and chatbot arena. In *Proc. of NeurIPS 2023*. New Orleans, LA, USA, (Dec. 2023).

[70] Kaijie Zhu et al. 2024. Promptrobust: towards evaluating the robustness of large language models on adversarial prompts. In *Proc. of LAMPS 2024*. Salt Lake City, UT, USA, (Oct. 2024).

[71] Yangtian Zi, Harshitha Menon, and Arjun Guha. 2025. More than a score: probing the impact of prompt specificity on LLM code generation. In *Proc. of IJCNLP 2025 and AACL 2025*. Mumbai, India, (Dec. 2025).