ISSN 1814-4225 (print)
ISSN 2663-2012 (online)

UDC 004.8:378.147:629.7. doi: 10.32620/reks.2026.2.11

Ihor TURKIN, Andriy CHUKHRAY, Oleksandr YEVDOKYMOV, Oleksandr NOSYKOV

National Aerospace University "Kharkiv Aviation Institute", Kharkiv, Ukraine

ADAPTIVE LEARNING OF FAULT-TOLERANT SOFTWARE ENGINEERING FOR UAV AND NANOSATELLITE ONBOARD COMPUTING

*This article **discusses** intelligent, computer-based learning methods for teaching reliability engineering principles to onboard computing platforms used in unmanned aerial vehicles and nanosatellites. This study focuses on integrating mathematical reliability models, simulation-based engineering tasks, and formal verification tools into an adaptive educational environment for safety-critical aerospace systems. **This study aims** to develop an educational methodology that combines adaptive intelligent tutoring with formal verification techniques to train students to design, analyze, and formally prove the correctness of reliability-related algorithms used in onboard computing systems. **The tasks** to be solved are: defining a competency model for fault-tolerant onboard computing, designing parameterized simulation-based learning tasks, integrating fault-injection scenarios into software quality assessment, and establishing correctness-critical computational kernels for adaptive tutoring. Additional tasks include developing a learner-modeling mechanism for individualized trajectories and validating reliability-related algorithms used in education. **The methods** used include: Monte-Carlo simulation of failure processes, analytic consistency checks, formal verification with Lean, Isabelle/HOL, and Rocq, fault-injection-driven software quality evaluation, and adaptive knowledge-tracing–based task sequencing. The proposed approach is demonstrated through an OnBoard1000 case study. The current validation is limited to a proof-of-concept prototype, reproducible computational checks, hidden-test diagnostics, and a prespecified classroom pilot protocol; completed classroom outcome data will be identified future work. **Conclusions**. The proposed approach demonstrates that combining simulation-driven engineering tasks with formal verification activities encourages students to validate the accuracy of their algorithms and reliability models rather than relying solely on numerical results. This supports the development of rigorous engineering reasoning required in safety-critical domains such as onboard computing in aerospace. Scientific novelty consists of integrating formal theorem-proving tools into an adaptive intelligent tutoring framework for reliability engineering education. The contribution is not a new knowledge-tracing algorithm but a domain-specific methodological layer that connects mission profiles and operating conditions, reliability assumptions and models, architectural redundancy decisions, and parameterized learning tasks with formally verified reference computations and adaptive task selection. In the proposed approach, formal proofs serve as a learning mechanism that teaches students to verify the correctness properties of algorithms and reliability models used in fault-tolerant onboard computing systems.*

Keywords*: intelligent tutoring system; adaptive learning; On Board Computer (OBC); On Board Diagnostics; reliability engineering; standby replacement; Monte-Carlo simulation; formal verification; Lean 4; Rocq; UAV; nanosatellite.*

1. Introduction

UAVs and nanosatellites increasingly rely on software-intensive onboard computers that must operate under mass, energy, cost, and maintenance constraints. Failures are unavoidable in such environments: radiation-induced soft errors, aging, thermal stress, and intermittent faults can degrade computation, sensors, and memory. Therefore, fault tolerance, becomes a mission-level requirement rather than an optional add-on, and engineers must be able to connect exact-science foundations (probability theory, reliability theory, discrete mathematics) with practical architectures (redundancy, voting, reconfiguration) and their algorithmic implementations [1].

Education in this domain is challenging for two main reasons. First, reliability and fault tolerance are inherently stochastic and strongly depend on the system architecture. Students benefit from simulation-based exploration and “what-if” analysis rather than static problem sets alone. Second, engineering tasks in reliability and fault-tolerant analysis often involve numerical models and algorithmic procedures whose correctness must be carefully validated. In safety-critical domains, such as aerospace computing, incorrect implementations or misunderstood models may lead to misleading conclusions about system behavior. Therefore, the educational process should not only teach how to compute reliability indicators but also require students to verify the correctness of the algorithms and models they use. This goal is supported by integrating formal verification tools into the learning process and cultivating the rigorous reasoning required for dependable onboard computing systems.

The study argues that combining adaptive, intelligent, computer-based learning [2] with formal verification techniques integrated into the educational process is

an effective solution. Adaptive tutoring enables individualized task selection and feedback, while theorem-proving systems such as Lean [3], Isabelle/HOL [4], and The Rocq Prover [5] allow students to formally analyze and prove the correctness properties of reliability models and algorithms. Such an approach encourages learners to implement computational procedures and verify their correctness and underlying assumptions. This study presents a methodology and platform model that integrates simulation-based reliability tasks with formal reasoning activities. It separates the implemented prototype components, proposed methodological extensions, and future classroom validation activities.

1.1. Motivation

The studies reported in this article lie at the intersection of machine learning and intelligent systems for education and dependable computing for aerospace cyber-physical systems. The practical context is the design and validation of fault-tolerant onboard computers, where failures may lead to mission loss, navigation degradation, or unsafe system behavior. For small UAVs and nanosatellites, limited hardware redundancy and limited maintenance opportunities make reliability-aware software, diagnostics, and graceful degradation important.

The gap between abstract mathematical models and engineering decisions is a recurring educational difficulty. Students often study probability distributions, expectations, and hazard rates in isolation but do not fully understand how these quantities influence architectural choices, such as redundancy structures (e.g., 1/2 vs 1/3 redundancy, TMR voting), parameter tuning, or system reconfiguration strategies. In terms of software reliability, there is no understanding that all software errors are the responsibility of the developing organization alone, since aging and degradation of software characteristics are completely absent under fixed conditions of use. Conversely, learners who focus mainly on programming may implement simulations without carefully analyzing the assumptions underlying the models, leading to conceptually incorrect results that appear plausible.

By embedding mathematical concepts in executable engineering tasks, intelligent computer-based learning can help bridge this gap. In such tasks, learners implement or complete simulation models, perform Monte-Carlo experiments, and compare simulation outcomes with reliability theory-derived analytic baselines. The learning environment provides stepwise scaffolding and adapts the task sequence based on the learner's progress and interaction patterns.

However, education in safety-critical engineering domains introduces an additional requirement: students must learn to compute reliability indicators and verify the correctness of the algorithms and models used to compute them. Reliability and fault-tolerant analysis is sensitive to modeling assumptions and numerical implementation details. Therefore, the proposed approach integrates formal verification activities into the tutoring process, encouraging students to analyze and prove the correctness properties of key computational procedures used in reliability modeling. In this paper, 'reliability-related algorithms" refers to computational procedures for calculating, simulating, and checking reliability indicators and fault-tolerance metrics. The phrase is descriptive and is not used as a standardized reliability term.

The OnBoard-1000 educational methodology developed at the National Aerospace University "Kharkiv Aviation Institute" provides a suitable foundation for this integration. It includes practical reliability engineering tasks based on fault injection modeling and hazard-rate interpretation. These elements naturally translate into learnable skills that combine mathematical reasoning, algorithmic implementation, and formal verification of correctness properties.

1.2. State of the art

The growing deployment of unmanned aerial vehicles (UAVs) in civil, industrial, and mission-critical applications has led to a continuous increase in the requirements for software reliability and fault tolerance. Modern UAVs operate in dynamic and uncertain environments, often performing autonomous or cooperative missions that require uninterrupted operation despite component failures, communication disruptions, and cybersecurity threats. Consequently, fault tolerance is no longer limited to handling isolated hardware malfunctions but is increasingly considered a system-wide property encompassing sensors, actuators, communication links, and onboard computing resources. For example, adaptive fault-tolerant control strategies have been proposed for tilt tri-rotor UAVs to maintain trajectory-tracking performance under servo faults, parametric uncertainties, and external disturbances [6]. Similar challenges arise in multi-agent systems, where coordinated operation must be maintained despite asynchronous topology switching, communication delays, and uncertain environmental conditions [7, 8]. These developments demonstrate that the software architectures of modern UAV systems must support increasingly sophisticated fault detection, diagnosis, recovery, and adaptive decision-making mechanisms.

A major trend in recent research is the application of artificial intelligence (AI) and machine learning (ML) to improve the reliability and resilience of UAV systems. Unlike traditional model-based approaches, AI-driven methods can learn complex relationships from opera-

tional data and adapt to changing conditions. Deep learning has been employed in sensor fusion to enable fault-tolerant state estimation without relying on hardware redundancy. Saied et al. proposed a framework that combines Long Short-Term Memory (LSTM) networks with fault detection mechanisms to maintain accurate UAV position and orientation estimation in the presence of sensor faults [9]. AI techniques are also being increasingly used to address cybersecurity challenges. The DroneGuard framework integrates supervised machine learning with explainable AI methods to detect GPS spoofing and denial-of-service attacks while providing interpretable explanations of the detection process [10]. Deep learning has been successfully applied to beam prediction and tracking for millimeter-wave communications in UAV communication systems, enabling reliable connectivity with reduced communication overhead [11]. Additionally, reinforcement learning approaches have been proposed to improve the resilience of robotic edge systems by minimizing recovery time and energy consumption following failures [12]. A comprehensive survey of UAV communications further highlights the growing importance of advanced ML paradigms, including federated learning, transfer learning, meta-learning, and explainable AI, for addressing emerging challenges in next-generation UAV networks [13]. Simultaneously, systematic reviews of UAV fuzzing techniques indicate that software defects and security vulnerabilities remain a significant concern and require more advanced automated testing and anomaly detection methods [14]. Table 1 summarizes the main distinctions between traditional reliability courses, general intelligent tutoring systems, and the proposed approach.

Table 1

Comparison with existing educational approaches

Feature	**Traditional instructor-led reliability courses**	**General-purpose intelligent tutoring systems**	**Proposed Approach**
Individualized adaptation of tasks	Possible, but primarily instructor-dependent and limited in scale	Automated for generic skills (programming, mathematics)	Automated for reliability-engineering skills (skill graph and knowledge tracing)
Creative engineering assignments (unconventional solutions, patent-literature analysis)	Established practice; formulated and assessed by the instructor	Rarely supported; requires substantial domain authoring	Not replaced; complemented by automated parameterized tasks
Domain-specific reliability task formulation (fault injection, redundancy, recovery)	Present in specialized courses; assessment is mainly manual	Typically absent from generic task banks	Integrated as parameterized task templates with automated checks
Automated learner-state tracking and task sequencing	Manual, based on instructor observation	Automated (knowledge tracing)	Automated and driven by domain-specific diagnostic checks
Automated diagnosis of reliability-engineering misconceptions	Depends on individual instructor feedback; hard to scale	Limited to generic error patterns	Diagnostic checks mapped to domain misconceptions (e.g., masked vs propagated faults)
Integration of mission profiles and operating conditions into tasks	Introduced through instructor-selected examples	Not typically represented	Defined as parameterized scenario configurations (UAV vs nanosatellite)
Formal or analytic validation of reference solutions	Manual checking by the instructor	Test-based, without formal guarantees	Analytic oracles and formally verified kernels (Lean 4)

The comparison in Table 1 characterizes each aspect's typical level of automation and domain integration rather than the principal capabilities of individual instructors or systems. Individually tailored assignments, creative tasks, and patent-literature analysis are well established in instructor-led reliability courses; therefore, the corresponding cells describe such elements as possible but primarily instructor-dependent, meaning that the instructor's time and experience rather than the teaching tradition itself limit their quality and scalability. Accordingly, the research gap addressed in this study is not the invention of adaptive or creative instruction as such, but the insufficient automation of these processes with advanced technologies, including AI – automated learner-state tracking, domain-specific generation of reliability tasks, and validated reference computations, for fault-tolerant embedded and onboard-system engineering.

The increasing complexity of fault-tolerant software systems creates new demands on engineering education. Existing educational standards and competency frameworks provide an essential foundation for software engineering curricula; however, they cannot fully capture the rapid technological evolution occurring in fields such as AI-driven reliability engineering, autonomous systems, and cybersecurity [15, 16]. As a result, educational programs must continuously adapt to emerging technologies and industrial requirements. Recent studies have demonstrated that generative AI can support curriculum

development by automating parts of the design process and providing content recommendations tailored to specific educational objectives [17]. Practical UAV-oriented educational initiatives emphasize integrating theoretical instruction with hands-on experimentation on real-world platforms and in engineering scenarios [18]. These observations suggest that preparing specialists capable of developing fault-tolerant onboard software requires educational approaches that extend beyond compliance with established standards and competency models.

A promising direction for addressing these challenges is adaptive learning. Intelligent tutoring systems have consistently demonstrated that personalized instruction, immediate feedback, and individualized task selection can significantly improve learning outcomes [19]. Learner modeling is a central component of such systems, which estimates a student's mastery of specific skills based on interactions with educational tasks. Knowledge Tracing [20, 21] remains one of the most influential approaches for modeling procedural knowledge acquisition and has become a foundational technique in adaptive educational systems. More recent developments have focused on scaling these methods to large educational datasets and parallelized implementations to improve the efficiency of learner analytics [22]. Reliability engineering education has traditionally emphasized probability distributions, survival functions, and fault-tree analysis as the primary means of teaching reliability assessment and system dependability [23]. Although these approaches provide a strong theoretical foundation, they rarely incorporate adaptive personalization or formal verification of computational procedures. This gap creates an opportunity to integrate adaptive tutoring, simulation-based reliability exercises, and formal verification techniques into reliability engineering education.

Unlike existing intelligent tutoring systems that primarily focus on programming, mathematics, or general STEM education, the proposed approach integrates reliability engineering simulation, fault-injection analysis, adaptive task sequencing, and formal theorem proving into a unified educational workflow for safety-critical aerospace software engineering. Compared with general learning management systems, code-runner environments, and conventional intelligent tutoring systems, the proposed approach is domain-specific: its task templates are built around onboard-computing reliability kernels, its feedback is tied to engineering misconceptions such as masked versus propagated faults, and its reference checks are linked to formally specified correctness properties. This distinction is important because the main educational risk in safety-critical software engineering is not only an incorrect final answer but also a plausible computation based on an invalid model or an unverified algorithm.

1.3. Objectives and tasks

This study aims to develop and justify intelligent computer-based learning methods for teaching fault-tolerant onboard computing in UAVs and nanosatellites, emphasizing the following:

– adaptive learning trajectories for exact-science reliability engineering foundations;

– formal verification of the reference algorithms used in the reliability modeling tasks.

To achieve the goal, within the framework of this study, the following **tasks** must be solved.

1. Define a domain-specific competency model for fault tolerance and reliability of onboard computers, including prerequisites between mathematical and engineering skills.

2. Design parameterized simulation-driven tasks and specify tutoring patterns for hints, feedback, and misconception diagnosis.

3. Propose a learner modeling and adaptation mechanism suitable for these tasks (skill graph + knowledge tracing) and describe how it drives task selection.

4. Identify correctness-critical computations in the tutoring loop and formulate a verification plan for theorem provers (Lean 4 (primary); Isabelle/HOL and The Rocq Prover (cross-validation)), including theorems and invariants that serve as trusted ``gold standards".

5. Demonstrate the approach on an OnBoard-1000-based case study and discuss evaluation metrics for learning outcomes and system trustworthiness.

The manuscript separates contributions into three layers. The implemented layer includes the OnBoard-1000 fault-injection task generator, reproducible simulation logs, analytic oracles, hidden tests, and diagnostic checks. The methodological layer includes the complete skill graph, adaptive sequencing policy, and proof-obligation catalog. The future work layer includes full classroom deployment, learner model parameter calibration on real student traces, and the expansion of the machine-checked proof artifact beyond the currently specified kernels.

The scientific novelty lies in:

– formalizing a portfolio of fault-tolerant onboard computing assessment kernels that are directly aligned with the engineering challenges;

– integrating these kernels into an adaptive learning loop as formally validated reference algorithms used in reliability modeling tasks and misconception detectors;

– adapting the workflow to UAV/nano-satellite mission profiles, resource constraints, and environmental stressors that shape both engineering decisions and learning errors.

Thus, the contribution of this study is not adaptive

ISSN 1814-4225 (print)
ISSN 2663-2012 (online)

instruction itself, nor is it a new formulation of the underlying knowledge-tracing algorithm, which is used in its standard Bayesian form. The proposed methodology contributes a domain-specific methodological layer that links mission profiles and operating conditions, reliability assumptions and hardware/software component models, architectural and redundancy decisions, parameterized learning tasks, formally or analytically validated reference solutions, reliability engineering misconception diagnosis, and adaptive task selection. The new theoretical elements are the formalization of this chain by the domain model (1)–(2) and the task tuples (3), the portfolio of correctness-critical assessment kernels with explicit proof obligations, and the use of formally verified reference computations as a trusted basis for diagnostic feedback in reliability engineering education.

A limitation at this stage is the lack of completed full-scale classroom data. To avoid overstating the evidence, the paper reports prototype-level computational validation, reproducible fault-injection scenarios, consistency checks between analytic and simulation results, and a prespecified evaluation protocol for future pilot deployment in undergraduate and graduate courses.

The article is structured as follows:

– section 2 describes the domain model, the adaptive tutoring workflow, and the verification-driven learning architecture.

– section 3 presents a case study based on the learning outcome of "fault injection & software quality."

– section 4 discusses the case study implications, limitations, comparison with existing educational platforms, and future work.

– section 5 summarizes the conclusions and the planned pilot classroom evaluation.

2. Materials and methods of research

2.1. Domain model and core skills

This study is based on a domain competency model for fault-tolerant onboard computing. The domain model is anchored in common onboard computing failure mechanisms (i.e., radiation-induced soft errors, intermittent faults, and aging) and their corresponding mitigation patterns (i.e., redundancy, voting, watchdogs, checkpoint/rollback, safe-mode reconfiguration). The mathematical foundations and architecture composition rules are based on the standard system reliability theory [23] and fault-tree analysis practice [24]. The parameter ranges and examples for electronics reliability prediction are aligned with the established reliability-prediction guidance [25, 26].

A structured domain model formalizes the relationships among competencies, learning outcomes, educational activities, and internationally recognized software engineering skill areas. This model provides the conceptual foundation for all subsequent analytical and computational steps, ensuring that the curriculum is both internally coherent and externally aligned with professional standards.

At the core of the domain model lie the following primary sets:

– C: - the set of general and professional competencies;

– L: - the set of learning outcomes (intended knowledge, skills, and abilities);

– A: - the set of educational activities (lectures and laboratory sessions);

– IL: - set of Indicators of Learning Outcomes (observable/measurable evidence of achieving outcomes);

– T-set of Learning Tasks (concrete activities assigned to learners);

– Spec - set of Key Properties / Specifications (defining features, constraints, or parameters of tasks)

– D: - set of Diagnostic Checks (assessment elements used to evaluate task performance);

– M is the set of Typical Misconceptions (common errors or incorrect mental models).

Each of these sets captures a distinct dimension of the intelligent computer-based learning methods for educational disciplines. Competencies represent the target capabilities that the course is intended to develop. Learning outcomes operationalize these competencies into measurable and observable results. Educational activities constitute the instructional mechanisms through which competencies and outcomes are achieved.

The model introduces two fundamental correspondence relations to establish a structured relationship among these sets:

– $R_1 \subseteq A \times C$, linking each activity to its competencies;

– $R_2 \subseteq A \times L$, linking each activity to the learning outcomes to which it contributes.

These relationships are weighted rather than binary, reflecting the varying degrees to which an activity contributes to different competencies or outcomes. This weighting captures the inherent multidimensionality of educational content: a single lecture may simultaneously reinforce architectural reasoning, standards compliance, and quality assurance principles to different extents

The subject area model is further extended by including mappings to software engineering skill areas (S) derived from the National Standards of Higher Education and international competency frameworks such as SWECOM (Software Engineering Competency Model). This introduces two additional relations:

– $f_1 \subseteq C \times S$, mapping competencies to one or more Skill Areas;

– $f_2 \subseteq L \times S$, mapping learning outcomes to Skill Areas.

These mappings indicate that competencies and learning outcomes often span multiple professional domains. For example, depending on the context and depth of coverage, architectural competencies in Software Engineering may relate to Software Design, Software Systems Engineering, and Software Safety simultaneously.

By integrating these mappings, the domain model forms the following multilayer structure:

$$\left.\begin{array}{l} A \xrightarrow{R_1} C \xrightarrow{f_1} S, \\ A \xrightarrow{R_2} L \xrightarrow{f_2} S. \end{array}\right\} \quad (1)$$

This structure enables the propagation of instructional relevance from activities to the corresponding Skill Areas through competencies and learning outcomes. The model supports a rigorous analysis of how instructional time is distributed across professional domains, ensuring alignment with industry expectations and accreditation requirements.

Thus, the domain model serves two critical purposes:

– semantic coherence-ensuring that competencies, outcomes, and activities are logically consistent;

– traceability-enabling transparent justification of how each instructional component contributes to the development of specific professional skills.

This formal foundation is essential for the subsequent quantitative stages of the methodology, including the construction of a weighted matrix, proportional allocation of instructional hours, and heatmap-based structural validation.

The construction of an adaptive learning system is carried out through the following mapping system:

– $g_1 \subseteq L \times IL$, linking each learning outcome to the Learning Outcome Indicators;

– $g_2 \subseteq IL \times T$ – each indicator is linked to multiple tasks that allow its assessment;

– $g_3 \subseteq T \times Spec$ – each task is characterized by a set of key properties or specifications;

– $g_4 \subseteq T \times D$ – each task includes multiple diagnostic checks to evaluate learner performance;

– $g_5 \subseteq D \times M$ – each diagnostic check is associated with typical misconceptions it is designed to detect.

As a result, we have a hierarchical mapping chain that provides a formal basis for implementing diagnostic assessment and supporting adaptive learning systems:

$$\left.\begin{array}{l} L \xrightarrow{g_1} IL \xrightarrow{g_2} T, \\ T \xrightarrow{g_3} Spec, \\ T \xrightarrow{g_4} D \xrightarrow{g_5} M. \end{array}\right\} \quad (2)$$

Thus, formula (1) connects the educational activities of the discipline, the acquired competencies and learning outcomes into one whole with the requirements of known standards and generally accepted recommendations, and system (2) allows you to directly move from learning outcomes to formulating the content of an adaptive learning system.

2.2. Task design and scaffolding patterns

Tasks in the proposed adaptive tutoring framework are formalized as parametric computational learning units that reflect the workflow of fault-tolerant software engineering: selecting a software fault model, implementing the corresponding metric-computation procedure, validating results through trusted consistency checks, and interpreting the outcome in terms of software robustness, detection, masking, propagation, and recovery behavior. Unlike traditional exercises, each task explicitly integrates domain concepts, executable computation, diagnostic feedback, and correctness validation.

Each task is defined as a tuple:

$$T = \langle M_f, P, K, V, R \rangle, \quad (3)$$

where M_f is a fault or software-behavior model;

P is a set of scenario parameters;

K is a computational kernel implemented or completed by the learner;

V is a set of validation constraints;

R is a reflection component requiring engineering interpretation of the result.

This formalization ensures that every task links theory, implementation, validation, and software-level engineering reasoning.

Tasks are decomposed into six cognitively ordered layers, each of which is associated with a specific type of reasoning. The learner recalls the distinction between fault, error, failure, masked fault, propagated fault, detected fault, false positive, and false negative at the concept activation layer. At the model-selection layer, the learner determines whether the scenario concerns detection coverage, error propagation, sensor deviation, or recovery behavior. At the parameter-instantiation layer, the learner configures the number of injections, the detection window, the threshold values, and the random seed. At the algorithm construction layer, the learner implements

ISSN 1814-4225 (print)
ISSN 2663-2012 (online)

log parsing and metric computation. At the validation layer, the learner checks the bounds, classification consistency, and recomputation from the raw logs. At the interpretation layer, the learner explains the implications of the computed indicators of the fault detection mechanism, safe-mode transition, or recovery strategy.

The task difficulty is increased by adding complexity to the fault-injection scenario: multiple fault locations, delayed detection, noisy sensors, recovery-time constraints, and mixed masked/propagated outcomes. Hints are gradually removed as mastery increases, and the learner moves from guided classification to independent implementation, validation, and interpretation of software fault-tolerance metrics.

Integration with formal verification is implemented by attaching reusable proof obligations to the computational kernels. These obligations include metric bounds, fault classification consistency, non-negativity of error counts, error-propagation matrices normalization, root mean square error and mean absolute error (RMSE/MAE) computation correctness, and bounded recovery-success estimation. The tutoring platform checks the properties automatically and may also be formalized as proof exercises in Lean, Isabelle/HOL, or Rocq for advanced learners.

Each implemented task template contains the following: (i) a machine-readable scenario configuration, (ii) a seed-controlled data generator, (iii) a learner-facing code or analysis stub, (iv) hidden validation tests, (v) a mapping from failed checks to misconceptions, and (vi) a reflection prompt. This structure makes explicit the boundary between an executable prototype component and the proposed instructional pattern.

2.3. Learner modeling and adaptation

The learner model maintains a probabilistic estimate of mastery for each skill node and updates it based on observable learner interactions, including responses to conceptual questions, intermediate step checks, code submission test results, time on task, and hint usage. To estimate the learner's current level of mastery, the proposed approach adopts a two-layer representation consisting of a prerequisite graph that encodes conceptual dependencies among skills and a probabilistic update mechanism based on knowledge-tracing models.

Knowledge tracing is used as a practical baseline approach: each skill is associated with a latent mastery state, and each learner interaction provides evidence of the learner's current mastery level. This evidence is richer than a single final numerical result in simulation-based tasks.

In the Bayesian Knowledge Tracing (BKT) model, each skill is associated with a mastery probability, along with a guess probability – q and a slip probability – s. The probability of a correct response is therefore:

$$P(C) = p_t(1-s) + (1-p_t)q. \quad (4)$$

After observing a correct response, the posterior estimate becomes the following:

$$p_t^+ = \frac{p_t(1-s)}{p_t(1-s)+(1-p_t)q}, \quad (5)$$

whereas after observing an incorrect response becomes

$$p_t^+ = \frac{p_t s}{p_t s+(1-p_t)(1-q)}. \quad (6)$$

The learning transition then updates mastery according to

$$p_{t+1} = p_t^+ + \left(1-p_t^+\right)p_{learn} \quad (7)$$

These equations are used to maintain interpretable estimates of skill mastery and support adaptive task sequencing in the tutoring system. Here, p_t is the current probability of skill mastery before the t-th interaction, p_t^+ is the posterior mastery probability after the observed response, s is the slip probability, q is the guess probability, and p_{learn} is the probability of transitioning from non-mastery to mastery after a learning opportunity.

The tutoring policy selects the next task to maximize expected learning gain while constraining cognitive load and learner frustration. When the learner struggles with skill S1, the platform provides shorter targeted tasks that isolate the relevant concept (e.g., deriving $R(t)$ from $F(t)$ and interpreting the parameter β). When the learner succeeds, the system's task complexity gradually increases, progressing from single-component reliability models to architecture-level aggregation and reconfiguration scenarios. The selection decision uses the updated mastery estimate p_{t+1}, failed diagnostic checks, hint use, and task-completion time: persistently low mastery or repeated failures trigger a simpler prerequisite task and additional hints, whereas stable mastery with successful validation advances the learner to a more complex task or the next skill. Here, $R(t)=1-F(t)$, $F(t)$ is the cumulative distribution function (CDF) of failure time, $R(t)$ is the corresponding reliability function, and β is the Weibull shape parameter.

The feedback follows a structured diagnose–explain–repair loop. The diagnostic stage uses a library of common misconceptions and counterexample tests. The explanation stage provides the minimal conceptual clarification required for progress. Finally, the repair stage suggests a concrete corrective action (e.g., editing a line of code, re-running a test, or answering a short verification question). This loop is aligned with the formally ver-

ified computational kernels described below, which provide reliable diagnostic checks, such as detecting CDF monotonicity violations.

2.4. Verification-driven tutoring and trusted reference kernels

The verification layer targets correctness-critical computations that shape feedback and solution evaluation in the tutoring process. The objective is not to formally verify the entire tutoring platform, but rather to prove small reusable computational kernels that appear in many tasks. These kernels serve as reference standards for validation, counterexample generation, and solution checking.

We prioritize properties that are both foundational and easy to implement incorrectly: bounds and monotonicity of CDF and reliability functions; correctness of architecture composition rules for non-repairable systems (e.g., series systems correspond to the minimum of component failure times); invariants of empirical CDF construction from samples (sorting, indexing, normalization); and no negativity and bounding of discrete mean time to failure (MTTF) estimators based on survival integration.

Interactive theorem provers are used to formalize and machine-check these properties. In this study, Lean 4 is treated as the primary prover for reference formalizations because it supports mathematical definitions close to textbook notation and can serve as a basis for proof-carrying specifications. Isabelle/HOL and The Rocq Prover are considered compatible alternatives for cross-validating selected lemmas and for future porting, rather than as simultaneous dependencies [3, 4, 5].

Verified kernels serve two complementary roles within the tutoring loop. First, they provide reliable reference computations for internal tests, thereby reducing the risk of systematic evaluation errors. Second, formalized properties can be used to generate targeted feedback.

2.5. Research protocol and reproducibility

The current study is presented as a design-oriented investigation supported by computational validation rather than a full classroom experiment. We implemented a proof-of-concept prototype as a lightweight Python-based simulation environment that executes parameterized reliability tasks, performs Monte-Carlo experiments, and validates learner solutions using reference computational kernels (analytic formulas and invariant-based checks). The numerical results reported in Section 3 were generated using this prototype. A classroom pilot study is planned within a bachelor-level course on reliability and fault-tolerant onboard computing, with a control group (traditional assignments) and an intelligent computer-based learning (ICBL) group.

The platform's learning analytics include task completion time, hint usage, code edits, and per-skill mastery traces derived from knowledge tracing models. In addition, a computational trust indicator, defined as the fraction of hidden test cases and property checks that are validated by formally verified kernels or cross-checked by analytic oracles, is introduced. This measure estimates the risk of undetected evaluation errors and complements educational metrics such as normalized learning gain and error-rate reduction.

The reproducibility is supported by:

– parameterized task templates with random explicit seeds;

– a mission-simulation configuration supporting MIL/SIL/PIL-style progression (model-in-the-loop, software-in-the-loop, processor-in-the-loop) from abstract models to embedded constraints; and

– reference implementations for failure-time generation, architecture aggregation, and reliability metric computation.

Figure 1 illustrates the proposed ICBL platform architecture. The adaptive tutor selects tasks from the content model, executes learner solutions in a simulation environment, and provides reliable reference computations for validation and feedback generation.

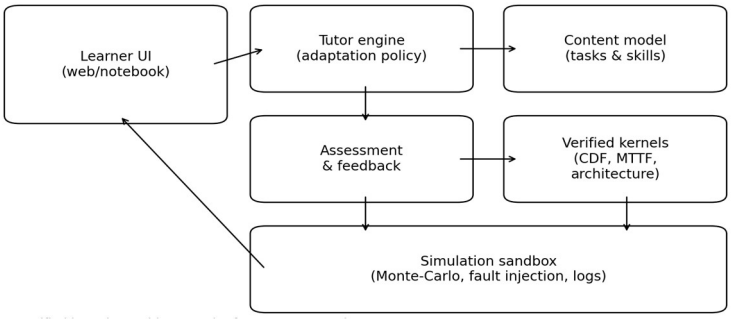

Fig. 1. Architecture of the proposed ICBL platform integrating simulation-based reliability tasks, adaptive tutoring, and formally verified reference kernels.

ISSN 1814-4225 (print)
ISSN 2663-2012 (online)

2.6. Validation Plan

Since the proposed platform is currently in the prototype stage, validation is organized as a multi-level process that evaluates both educational effectiveness and computational trustworthiness.

The validation strategy comprises four complementary levels.

Level 1. Computational validation

The first level verifies the correctness of the educational kernels used for task generation and assessment.

The following checks are performed:

- agreement between the analytical reliability models and Monte-Carlo simulation results;
- property-based testing of the reliability metrics
- verification of the mathematical invariants;
- cross-validation between the independent implementations;
- formal proof checking in Lean 4 and the selected Isabelle/HOL and Rocq models.

The primary quantitative indicator is the computational trust index:

$$CTI = \frac{N_{verified}}{N_{total}} \quad (8)$$

where $N_{verified}$ denotes the number of assessment kernels validated through formal verification or independent analytical confirmation. The denominator N_{total} denotes the total number of assessment kernels included in the validation set.

Level 2. Expert validation

A panel of experts in software engineering, reliability engineering, and aerospace computing evaluates the following:

- relevance of learning outcomes;
- realism of fault-injection scenarios;
- quality of adaptive feedback;
- correctness of competency mappings.

Evaluation is performed using a five-point Likert scale.

Level 3. Pilot educational study

The platform will be evaluated in undergraduate and graduate software engineering courses.

Two groups are planned:

- Control Group (traditional laboratory assignments);
- Experimental Group (adaptive intelligent tutoring platform).

The study duration is expected to cover one academic module (4–6 weeks).

Measured variables include the following:

- pre-test and post-test scores;
- normalized learning gain;
- task completion time;
- error rate;
- number of hints used;
- retention of knowledge after delayed testing.

The normalized learning gain is computed as

$$g = \frac{Post - Pre}{100 - Pre} \quad (9)$$

where Pre and Post denote the pretest and posttest scores, respectively.

Level 4. User acceptance evaluation

Student perceptions will be collected using questionnaires based on the Technology Acceptance Model (TAM).

The following factors will be measured:

- perceived usefulness;
- perceived ease of use;
- learning motivation;
- confidence in automated assessment;
- perceived value of formal verification activities.

Success criteria

The platform is considered successful if:

- normalized learning gain exceeds that of the control group by at least 15%;
- error rates decrease significantly ($p < 0.05$);
- expert evaluation exceeds 4.0/5.0;
- computational trust index exceeds 0.95;
- student satisfaction exceeds 80%.

The proposed validation framework enables future empirical evaluation of both educational effectiveness and correctness assurance mechanisms and provides a reproducible basis for subsequent large-scale studies.

3. Results

The case study is based on the OnBoard-1000 educational methodology developed at National Aerospace University "Kharkiv Aviation Institute", which provides laboratory materials for the analysis of reliability in onboard computing systems. The described models are implemented as reusable platform components, including a task generator, a simulation runner, and a set of formally verified kernels.

The discipline “UAV and Nanosatellite Onboard Computing” involves achieving the following learning outcomes:

– understanding of standards and recommendations relevant to space systems and flight software;

– ability to analyze information flows, telemetry, events, and command structures in embedded and aerospace systems;

– ability to formulate functional and non-functional requirements for software components and subsystems;

– ability to analyze architectural decisions, evaluate alternatives, and justify system-level design choices;

– ability to assess software quality, reliability, and robustness using appropriate methods and tools;

– ability to design, modify, and integrate software architecture components within complex embedded systems.

The learning outcome "Ability to assess software quality, reliability, and robustness using appropriate methods and tools" includes, among others, the tutoring kernel "Faults injection":

– fault coverage computation – assessment of the system's ability to detect injected errors using formal metrics (coverage, detection rate, false positive/negative results);

– error propagation assessment when faults are injected – calculation of conditional probabilities; construction of matrices of system errors and failures depending on injected faults; estimation of the probability of failure;

– sensor error modeling – application of noise/bias/drift models; calculation of deviation metrics (RMSE/MAE); threshold violation detection;

– assessment of recovery behavior – calculation of recovery time; estimation of limited recovery; estimation of the probability of success.

The learning task families in Table 2 were developed in two steps. First, the hierarchical mapping chain (2) was applied to the discipline's learning outcomes: each outcome was decomposed into indicators, and task families were specified as tuples (3) with key properties, diagnostic checks, and typical misconceptions for each indicator. Second, the content of the task families follows the engineering reasoning sequence used in the reliability assessment of onboard systems [23, 24, 25, 26]: from the mission profile and operating conditions, which differ between UAVs and nanosatellites, through the justification of the time-to-failure distribution law and its underlying assumptions and the account of environmental impact factors on hardware failure rates, to the selection of reliability models for hardware, software, and field-programmable gate array (FPGA) components, the construction of a reliability block diagram (RBD) from architectural analysis, and the analysis and selection of redundancy methods for software and the system as a whole.

Table 2 summarizes the mapping between tutoring kernels, typical learning tasks, proof obligations, and automated diagnostic strategies. This mapping also functions as a roadmap for incremental deployment: the platform can initially rely on analytic consistency checks and progressively replace them with formally verified implementations. Table 2 presents a representative rather than exhaustive set of task families: the four fault-injection families are implemented in the current prototype (Section 2.5), whereas the families marked with an asterisk (*) are defined as methodological task templates within formalization (3) and are intended for the planned pilot deployment.

Table 2

Representative learning task families, their key properties, diagnostic checks, and typical misconceptions

Task family / kernel	**Learning task**	**Key property / specification**	**Diagnostic checks**	**Typical misconception**
Fault coverage computation	Classify injected faults; compute coverage, detection rate, FP/FN	$0 \le C_F \le 1$; $C_F = \frac{N_d}{N_i}$; consistency: $N_d + N_u + N_m = N_i$	Coverage formula check; classification consistency; FP/FN bounds; recomputation from raw logs	Confusing coverage with detection rate; treating masked faults as detected; ignoring false positives; assuming coverage=1 means perfect reliability
Error propagation under injection	Compute conditional probabilities; build error→failure matrix; estimate failure likelihood	$0 \le P(E \to F) \le 1$; row sums = 1; monotonicity of failure probability	Matrix normalization; conditional probability checks; cross-checking transitions with logs	Mixing unconditional and conditional probabilities; misinterpreting masked errors; assuming every error leads to failure
Sensor fault simulation	Apply noise/bias/drift models; compute deviation metrics (RMSE/MAE); detect threshold violations	$RMSE \ge 0$; threshold crossing logic; noise model consistency	RMSE/MAE recomputation; bounds checks; noise model parameter validation	Confusing noise with bias; ignoring drift; assuming any deviation is a fault; incorrect RMSE normalization
Recovery behavior evaluation	Compute recovery time; evaluate bounded	$T_{rec} \le T_{max}$; $0 \le P_{rec} \le 1$; stable-state convergence	Recovery time measurement; convergence detection; probability	Assuming watchdog "fixes" errors; confusing restart

	recovery; estimate success probability		estimation consistency	with recovery; ignoring unstable post-recovery oscillations
Time-to-failure model selection*	Justify the time-to-failure distribution law and its assumptions from the mission profile and operating conditions (UAV vs nanosatellite)	R(t) = 1 − F(t); assumptions consistent with the declared operating conditions	Assumption checklist; analytic vs Monte-Carlo consistency	Adopting a constant hazard rate by default; transferring UAV operating assumptions to satellite missions
Environmental impact on HW failure rates*	Account for environmental impact factors on component failure rates; compute HW reliability of the onboard computer	Adjusted failure rates non-negative; parameter ranges aligned with reliability-prediction guidance [25, 26]	Range checks of adjusted failure rates; recomputation of system-level reliability	Using laboratory (ground-benign) failure rates for flight environments
Reliability model selection for SW and FPGA components*	Select and justify reliability models for software and FPGA-based components as distinct from hardware models	Software failures treated as design faults; no aging under fixed conditions of use	Model-choice justification against declared fault classes	Attributing aging or degradation to software; applying hardware failure-rate models to software
RBD construction and redundancy requirements*	Construct an RBD from architectural analysis; identify and justify redundancy requirements	Series composition corresponds to the minimum of component failure times; RBD consistent with the architecture	Structure consistency check; analytic vs simulation comparison	Confusing series and parallel structures; expecting redundancy to help regardless of voting and switching overhead
Redundancy method selection for SW and system*	Analyze and select redundancy methods (e.g., TMR voting, standby replacement) for software and the system as a whole	Alternatives compared under identical mission constraints	Scenario-based simulation comparison of alternatives	Expecting identical redundant software copies to tolerate their own design faults

The following example illustrates a typical programming-and-validation task derived from the OnBoard-1000 methodological materials and implemented in the proposed ICBL platform.

Context: a simplified attitude control subsystem (ACS) runs on an onboard computer and periodically computes actuator commands based on sensor data and internal state. A software fault detection mechanism (FDM) monitors the ACS outputs and internal variables, raising an alarm flag and switching the system to a safe mode when an anomaly is detected.

The goal of the tutoring task is to estimate and validate the fault coverage of the FDM using a controlled fault-injection experiment.

System and fault detection model. The ACS control loop is run with a fixed period T_s. The FDM can:

– raise an alarm (alarm = 1);

– leave the system in the nominal mode (alarm = 0);

– or be bypassed (for baseline runs).

Fault model for this example:

– single-bit flips in the selected state variables and intermediate computations;

– injected at the predefined time steps k;

– affecting either internal state x_k or control output u_k.

Each injected fault is classified (by the ground truth) as:

– masked - no deviation at the system output and no safety violation,

– propagated - deviation at the system output or safety bound violation,

– detected - FDM alarm raised in response to the fault.

Example parameters. For demonstration, the following parameters are used:

– number of injection points per run: $n_{inj} = 50$,

– number of Monte-Carlo runs: $N = 1000$,

– total number of injected faults: $N_i = N \cdot n_{inj}$,

– random seed = 123 for reproducibility.

In the tutoring platform these values are automatically parameterized for individual learners.

Fault injection procedure for each Monte-Carlo run.

1.The ACS is initialized in a nominal state and executed for a fixed horizon.

2.A fault is injected at each predefined injection point according to the fault model (bit-flip in a selected variable).

3.For each injected fault, the following are recorded:

– whether the FDM raised an alarm within a specified detection window,

– whether the system output violated a predefined safety bound,

– whether the deviation remained within the acceptable limits (masked fault).

The learner receives a log containing, for each injected fault:

– injection ID;

– injection time step;

– fault location/type;

– detected flag;

– failure flag (safety violation);

– masked flag.

From the log, the learner must compute:

– total number of injected faults N_i;

– number of detected faults N_d;

– number of undetected but propagated faults N_u;

– number of masked faults N_m;

– number of false positives N_{FP} (alarms without injected fault in the window);

– number of false negatives N_{FN} (propagated faults without alarm).

The fault coverage is defined as:

$$C_F = \frac{N_d}{N_i} \quad (10)$$

with the following consistency constraint:

$$N_d + N_u + N_m = N_i. \quad (11)$$

Optionally, the learner also computes the following:

– detection rate:

$$DR = \frac{N_d}{N_d + N_u}, \quad (12)$$

– false positive rate:

$$FPR = \frac{N_{FP}}{N_{FP} + N_{TN}}, \quad (13)$$

where N_{TN} is the number of true negatives in non-faulty intervals.

Validation and acceptance criteria. The tutoring platform applies several automated validation checks as follows:

– range checks: $0 \le C_F \le 1$, $0 \le DR \le 1$;

– consistency check: verify that

$$N_d + N_u + N_m = N_i; \quad (14)$$

– classification sanity: no fault is simultaneously marked as both masked and failure-inducing; no false positive is counted as detected fault;

– scenario-based tolerance: the computed coverage C_F is compared against a reference value $C_{F,ref}$ obtained from a trusted baseline implementation. The result is considered consistent if $|C_F - C_{F,ref}|$ lies within a predefined tolerance derived from the campaign size.

Within the tutoring platform, these checks are implemented as trusted computational kernels that:

– recompute the metrics from the raw log;

– detect typical student errors (e.g., misclassification, wrong formula, missing normalization);

– and generate diagnostic feedback when inconsistencies are found.

Thus, this example operationalizes the Fault coverage computation kernel as a concrete, parameterized task suitable for adaptive learning.

4. Discussion

This case study suggests several design principles for STEM tutoring in safety-critical engineering. First, simulation-based tasks are particularly effective when paired with analytic baselines that learners can compute and interpret. This pairing transforms validation into an explicit learning objective and reduces the risk that learners will accept plausible but incorrect computational results.

Second, adaptivity is most effective when it is skill-aware. A learner who misinterprets the parameter β should not proceed to architecture-level tasks immediately. Conversely, a learner who consistently passes analytic checks but fails implementation tests typically requires additional programming scaffolding rather than a more theoretical explanation. A prerequisite graph combined with knowledge tracing provides a practical mechanism for such adaptive decisions without the need for large training datasets.

Third, the correctness of reference computations is critical in probabilistic and numerical domains. Even small errors in the underlying algorithms may propagate

ISSN 1814-4225 (print)
ISSN 2663-2012 (online)

through automated learning systems, distorting conceptual understanding. Formally verified kernels mitigate this risk and provide transparent explanations through invariants and constructive counterexamples, which improves feedback reliability and interpretability.

From an engineering perspective, the proposed tasks reflect the reasoning process used in real onboard system development: selecting reliability models, validating assumptions, simulating failure processes, and evaluating redundancy and reconfiguration strategies. Therefore, learners practice engineering argumentation, explaining not only the computed value but also why the result is credible and what it implies for a design decision.

Future work includes integrating the tutoring sandbox with hardware-in-the-loop experiments, introducing controlled fault-injection scenarios in emulated processors, and extending formal verification from mathematical kernels to selected components of student-written code through certified checking mechanisms. These directions may further strengthen the connection between reliability engineering education and trustworthy onboard computing systems.

The proposed validation framework addresses one of the major challenges in intelligent engineering education: demonstrating not only learning effectiveness but also the trustworthiness of automated assessment. The framework provides a comprehensive methodology for future empirical assessment of the platform by combining computational validation, expert review, pilot classroom studies, and user acceptance evaluation.

4. Conclusions

This article presented intelligent computer-based learning methods for teaching fault-tolerant onboard computing in UAVs and nanosatellites, combining simulation-driven reliability tasks with adaptive tutoring and a formal verification layer for correctness-critical reference computations.

To ensure reliability and fault tolerance, a domain competency model was defined that links exact-science foundations to architecture-level design decisions and reconfiguration logic. A task design methodology based on tutoring kernels and diagnostic feedback for Fault Injection and Software Quality was proposed, enabling training of both computational skills and engineering thinking. This study describes a learner modeling and adaptation approach (skill graph, knowledge tracing, and misconception diagnostics) that enables individualized learning trajectories and targeted feedback for common errors in reliability modeling tasks. Taken together, these elements constitute the main scientific contribution: a domain-specific methodological layer that connects mission profiles and operating conditions, reliability assumptions and component models, architectural and redundancy decisions, and parameterized learning tasks with formally verified reference computations and adaptive selection of the subsequent task; the underlying knowledge-tracing algorithm is used in its standard form.

A verification roadmap was developed to integrate theorem provers (Lean, Isabelle/HOL, The Rocq Prover) as sources of formally verified reference algorithms implemented via small, reusable kernels. This approach reduces the risk of incorrect computations in safety-critical reliability education.

Future work will focus on pilot classroom evaluation, extending the library of verified kernels, and integrating formal proof artifacts and feedback generation (i.e., proof-guided hints and certified counterexamples).

In addition, a multi-level validation framework was proposed that covers computational correctness, expert evaluation, educational effectiveness, and user acceptance. This framework provides a foundation for future empirical studies and large-scale deployment of the proposed adaptive learning environment.

Contribution of authors: domain analysis (reliability and fault tolerance), case study design, conceptualization, writing – review and editing, supervision: **Ihor Turkin**; original draft preparation, verification roadmap, simulation: **Oleksandr Yevdokymov**. tutoring model design, task design and scaffolding patterns: **Andriy Chukhray,** abstract, conclusions – **Oleksandr Nosykov**; design of the article – **Oleksandr Yevdokymov, Oleksandr Nosykov.**

All authors have read and approved the final manuscript for publication.

Conflict of Interest

The authors declare that they have no conflicts of interest related to this research, whether financial, personal, authorship, or otherwise, that could affect the research or the results presented in this paper.

Financing

This research is carried out at the expense of the general fund of the state budget, "Methods and tools of ensuring fault tolerance of on-board hardware and software systems of UAV and nanosatellite computing platforms". State registration number: 0126U002262

Data Availability

The manuscript has no associated data.

Use of Artificial Intelligence

The authors have used artificial intelligence technologies within acceptable limits to support drafting and language editing of the manuscript. All models, algo-

rithms, numerical examples, and conclusions were verified and approved by the authors in accordance with the research methodology described in the paper.

References

1. Bertot, Y., & Castéran, P. Interactive Theorem Proving and Program Development: Coq'Art. Berlin: Springer, 2004, 469 p. DOI: 10.1007/978-3-662-07964-5.
2. CADFEM. Reliability analysis for electronic/electrical systems. *CADFEM Engineering Services*, 2026. Available at: https://www.cadfem.net (accessed 31 May 2026).
3. Corbett, A. T., & Anderson, J. R. Knowledge tracing: Modeling the acquisition of procedural knowledge. *User Modeling and User-Adapted Interaction*, 1994, vol. 4, no. 4, pp. 253–278. DOI: 10.1007/BF01099821.
4. Das Adhikary, P., & Metsämuuronen, J. Knowledge tracing models in digital learning: Historical evolution, categorization, and empirical evaluation. *ResearchGate preprint*, 2025. DOI: 10.13140/RG.2.2.17277.27364.
5. De Moura, L., & Ullrich, S. The Lean 4 theorem prover and programming language. *Automated Deduction – CADE 28, Lecture Notes in Computer Science*, Cham: Springer, 2021, vol. 12699, pp. 625–635. DOI: 10.1007/978-3-030-79876-5_37.
6. European Space Agency. System level methods. *Digital Reliability Handbook*, 2026. Available at: https://handbook.reliability.space/en/latest/system/handbook/reliability_prediction/methods.html (accessed 31 May 2026).
7. Hao, W., Xian, B., & Xie, T. Fault-tolerant position tracking control design for a tilt tri-rotor unmanned aerial vehicle. *IEEE Transactions on Industrial Electronics*, 2022, vol. 69, no. 1, pp. 604–612. DOI: 10.1109/TIE.2021.3050384.
8. Hess, A. V., Mödersheim, S. A., & Brucker, A. D. Stateful protocol composition in Isabelle/HOL. *ACM Transactions on Privacy and Security*, 2023, vol. 26, no. 3, article no. 25. DOI: 10.1145/3577020.
9. Hong, J.-H., Shin, H.-S., & Tsourdos, A. A design of a short course with COTS UAV system for higher education students. *IFAC-PapersOnLine*, 2019, vol. 52, no. 12, pp. 466–471. DOI: 10.1016/j.ifacol.2019.11.287.
10. Hussein, M., & Nouacer, R. Reference architecture specification for drone systems. *Microprocessors and Microsystems*, 2022, vol. 95, article no. 104705. DOI: 10.1016/j.micpro.2022.104705.
11. Ihekoronye, V. U., Ajakwe, S. O., Lee, J. M., & Kim, D.-S. DroneGuard: An explainable and efficient machine learning framework for intrusion detection in drone networks. *IEEE Internet of Things Journal*, 2025, vol. 12, no. 7, pp. 7708–7722. DOI: 10.1109/JIOT.2024.3519633.
12. Impagliazzo, J., Bourque, P., & Mead, N. R. Incorporating CC2020 and SWECOM competencies into software engineering curricula: A tutorial. *Proceedings of the IEEE 32nd Conference on Software Engineering Education and Training (CSEE&T)*, Munich, Germany, 2020, pp. 1–3. DOI: 10.1109/CSEET49119.2020.9206238.
13. Kurup, L. D., Joshi, A., & Shekhokar, N. A review on student modeling approaches in ITS. *Proceedings of the 3rd International Conference on Computing for Sustainable Global Development (INDIACom)*, New Delhi, India, 2016, pp. 2513–2517.
14. Li, S., Jin, J., Afrin, M., Zheng, Q., Fu, J., & Tian, Y.-C. UAV-as-a-Service for robotic edge system resilience. *Proceedings of the IEEE International Conference on Web Services (ICWS)*, Shenzhen, China, 2024, pp. 142–148. DOI: 10.1109/ICWS62655.2024.00034.
15. Malviya, V. K., Minn, W., Shar, L. K., & Jiang, L. Fuzzing drones for anomaly detection: A systematic literature review. *Computers & Security*, 2025, vol. 148, article no. 104157. DOI: 10.1016/j.cose.2024.104157.
16. Nugroho, V. A., & Lee, B. M. GPS-aided deep learning for beam prediction and tracking in UAV mmWave communication. *IEEE Access*, 2025, vol. 13, pp. 117065–117077. DOI: 10.1109/ACCESS.2025.3586594.
17. Paramesha, K., Hamsaveni, M., & Soumya, B. J. Artificial intelligence driven curriculum development: Challenges and modalities. *Proceedings of the Annual International Conference on Data Science, Machine Learning and Blockchain Technology (AICDMB)*, Mysuru, India, 2025, pp. 1–7. DOI: 10.1109/AICDMB64359.2025.11277952.
18. Pu, Y., Wu, W., Han, Y., & Chen, D. Parallelizing Bayesian knowledge tracing tool for large-scale online learning analytics. *Proceedings of the IEEE International Conference on Big Data (Big Data)*, Seattle, WA, USA, 2018, pp. 3245–3254. DOI: 10.1109/BigData.2018.8622355.
19. Ramezanian, S., & Niemi, V. Cybersecurity education in universities: A comprehensive guide to curriculum development. *IEEE Access*, 2024, vol. 12, pp. 61741–61766. DOI: 10.1109/ACCESS.2024.3392970.
20. Rausand, M., & Høyland, A. System Reliability Theory: Models, Statistical Methods, and Applications. 2nd ed. Hoboken, NJ: Wiley, 2003, 644 p. DOI: 10.1002/9780470317046.
21. Saied, M., Mishi, A., Francis, C., & Noun, Z. A deep learning approach for fault-tolerant data fusion applied to UAV position and orientation estimation. *Electronics*, 2024, vol. 13, no. 16, article no. 3342. DOI: 10.3390/electronics13163342.
22. Shi, Y., Li, J., Lv, M., Wang, N., & Zhang, B. Distributed consensus control for 6-DOF fixed-wing multi-UAVs in asynchronously switching topologies. *IEEE Transactions on Vehicular Technology*, 2025, vol. 74, no. 4, pp. 5649–5663. DOI: 10.1109/TVT.2024.3520141.
23. Sun, C., et al. Advancing UAV communications: A comprehensive survey of cutting-edge machine learning techniques. *IEEE Open Journal of the Vehicular Technology*, 2024, vol. 5, pp. 825–854. DOI: 10.1109/OJVT.2024.3401024.
24. Tolo, S., & Andrews, J. Fault tree analysis including component dependencies. *IEEE Transactions on Reliability*, 2024, vol. 73, no. 1, pp. 413–421. DOI: 10.1109/TR.2023.3264943.
25. VanLehn, K. The behavior of tutoring systems. *International Journal of Artificial Intelligence in Education*, 2006, vol. 16, no. 3, pp. 227–265. DOI: 10.3233/IRG-2006-16(3)02.
26. Zhang, K., Zhang, W., Du, X., & Li, Z. Fixed-time event-triggered sliding mode consensus control for multi-AUV formation under external disturbances and communication delays. *Journal of Marine Science and Engineering*, 2025, vol. 13, no. 12, article no. 2294. DOI: 10.3390/jmse13122294.

Received 11.05.2026, Received in revised form 15.06.2026
Accepted date 18.07.2026, Published date 31.07.2026

ISSN 1814-4225 (print)
ISSN 2663-2012 (online)

ІНТЕЛЕКТУАЛЬНІ МЕТОДИ КОМП'ЮТЕРНОГО НАВЧАННЯ ДЛЯ ВИКЛАДАННЯ ВІДМОВОСТІЙКИХ БОРТОВИХ ОБЧИСЛЮВАЧІВ БПЛА ТА НАНОСУПУТНИКІВ

І. Б. Туркін, А. Г. Чухрай, О. О. Євдокимов, О. С. Носиков

Предметом статті є інтелектуальні комп'ютерні методи навчання для викладання принципів інженерії надійності бортових обчислювальних платформ, що використовуються в безпілотних літальних апаратах та наносупутниках. Дослідження зосереджено на інтеграції математичних моделей надійності, інженерних завдань на основі моделювання та інструментів формальної верифікації в адаптивне освітнє середовище для критично важливих для безпеки аерокосмічних систем. Метою дослідження є розробка освітньої методології, яка поєднує адаптивне інтелектуальне навчання з методами формальної верифікації, щоб навчити студентів проєктувати, аналізувати та формально доводити правильність алгоритмів, пов'язаних із надійністю, що використовуються в бортових обчислювальних системах. Завдання, які необхідно вирішити: визначення моделі компетентності для відмовостійких бортових обчислень, проєктування параметризованих навчальних завдань на основі моделювання, інтеграція сценаріїв впровадження несправностей в оцінку якості програмного забезпечення та встановлення критичних для правильності обчислювальних ядер для адаптивного навчання. Додаткові завдання включають розробку механізму моделювання учня для індивідуалізованих траєкторій та валідацію алгоритмів, пов'язаних із надійністю, що використовуються в освіті. Використані методи включають: моделювання процесів відмов методом Монте-Карло, аналітичні перевірки на узгодженість, формальну верифікацію за допомогою Lean, Isabelle/HOL та Rocq, оцінку якості програмного забезпечення на основі впровадження несправностей, а також адаптивне послідовне визначення порядку завдань на основі трасування знань. Підхід демонструється на прикладі OnBoard 1000. Висновки. Запропонований підхід демонструє, що поєднання інженерних завдань, заснованих на моделюванні, з формальними верифікаційними заходами заохочує студентів перевіряти правильність своїх алгоритмів і моделей надійності, а не покладатися виключно на числові результати. Це сприяє розвитку ретельного інженерного мислення, необхідного в критично важливих для безпеки галузях, таких як бортові обчислення в аерокосмічній галузі. Наукова новизна полягає в інтеграції формальних інструментів доведення теорем в адаптивну інтелектуальну навчальну основу для навчання з питань надійності. Внесок полягає не в новому алгоритмі трасування знань, а в предметно-орієнтованому методичному рівні, який пов'язує профілі місій та умови експлуатації, припущення і моделі надійності, архітектурні рішення щодо резервування та параметризовані навчальні завдання з формально верифікованими еталонними обчисленнями й адаптивним вибором наступного завдання. У запропонованому підході формальні докази використовуються як механізм навчання, який навчає студентів перевіряти властивості правильності алгоритмів і моделей надійності, що використовуються у відмовостійких бортових обчислювальних системах.

Ключові слова: *інтелектуальна навчальна система; адаптивне навчання; бортовий комп'ютер; бортова діагностика; інженерія надійності; k-з-n резервування; TMR 2/3; резервування із заміщенням; метод Монте-Карло; формальна верифікація; Lean; БПЛА; наносупутник.*

Туркін Ігор Борисович – д-р техн. наук, проф., зав. кафедри інженерії програмного забезпечення, Національний аерокосмічний університет «Харківський авіаційний інститут», Харків, Україна.

Чухрай Андрій Григорович – д-р техн. наук, проф., проф. кафедри інженерії програмного забезпечення, Національний аерокосмічний університет «Харківський авіаційний інститут», Харків, Україна.

Євдокимов Олександр Олегович – магістр, аспірант кафедри математичного моделювання та штучного інтелекту, Національний аерокосмічний університет «ХАІ», Харків, Україна.

Носиков Олександр Сергійович – магістр, аспірант кафедри інженерії програмного забезпечення, Національний аерокосмічний університет «Харківський авіаційний інститут», Харків, Україна.

Ihor Turkin – Dr. of Sciences, Professor. Head of Software Engineering Department, National Aerospace University "Kharkiv Aviation Institute", Kharkiv, Ukraine.
e-mail: i.turkin@khai.edu, ORCID: 0000-0002-3986-4186, Scopus Author ID: 57203145725.

Andriy Chukhray – Doctor of Technical Sciences, Professor, Professor of the Department of Software Engineering, National Aerospace University "Kharkiv aviation institute", Kharkiv, Ukraine,
e-mail: a.chukhray@khai.edu, ORCID: 0000-0002-8075-3664.

Oleksandr Yevdokymov – Master, PhD Student of the Department of Mathematical Modeling and Artificial Intelligence, National Aerospace University "Kharkiv Aviation Institute", Kharkiv, Ukraine.
e-mail: o.yevdokymov@khai.edu, ORCID: 0009-0008-9687-6344, Scopus Author ID: 58099243700.

Oleksandr Nosykov – Master, PhD Student Department of Software Engineering, National Aerospace University "Kharkiv Aviation Institute", Kharkiv, Ukraine.
e-mail: o.nosykov@khai.edu, ORCID: 0000-0002-3233-5636.