NUANSA INFORMATIKA
Volume 20 Nomor 01, Januari 2026

p-ISSN :1858-3911* , *e-ISSN : 2614-5405
https://journal.fkom.uniku.ac.id/ilkom

Real-Time Bayesian Knowledge Tracing for Adaptive Vocabulary Practice in an Adventure Educational Game: Architecture, Verification, and Reproducibility

Ikhsan Khaeruddin[1], Rio Andriyat Krisdiawan*[2], Nida Amalia Nasikin[3]
[123]*Teknik Informatika, Fakultas Ilmu Komputer Universitas Kuningan, Indonesia*
E-mail: [1]20210810111@uniku.ac.id, *[2]rioandriyat@uniku.ac.id,[3]nida.amalia.asikin@uniku.ac.id

Abstract

Vocabulary learning in primary English classes is often constrained by heterogeneous learner readiness, where fixed game progression can under-serve both struggling and advanced students. This study aims to implement and technically validate a reproducible real-time adaptivity mechanism using Bayesian Knowledge Tracing (BKT) in a Unity-based Android vocabulary game. The research adopts a design-and-verification approach using the Game Development Life Cycle (GDLC), supported by requirements elicitation (classroom observation, teacher interview, and literature review). The adaptive engine applies BKT to update mastery after each quiz response and routes learners using a mastery-threshold policy, while event-level logs are stored locally and exportable for auditability. The main results demonstrate that adaptive mode activation, mastery updates, persistence of adaptive state, and mastery-gated progression function consistently in end-to-end black-box tests. Algorithm-level credibility is strengthened through white-box basis-path verification of the UpdateProbability() routine, ensuring independent execution paths for correct and incorrect responses are covered. This work contributes a deployable Unity/Android architecture for real-time BKT-driven adaptivity, accompanied by verification artifacts and reproducibility recommendations to support technical audit, replication, and subsequent controlled effectiveness studies.

Keywords— *Bayesian Knowledge Tracing (BKT), Adaptive learning, Educational game, GDLC, Software verification.*

Submission: December 08, 2025 Accepted: Jauary 20, 2026 Published: January 20, 2026
DOI Article: ***https://doi.org/10.25134/ilkom.v20i1.535***

1. INTRODUCTION

Vocabulary knowledge is a fundamental constraint in early English learning; limited lexical coverage quickly becomes a bottleneck for comprehension and classroom participation [1]. Digital game-based language learning has expanded because games can increase engagement and provide repeated exposure, and meta-analytic evidence indicates that game-based approaches can yield measurable learning benefits when aligned with instructional goals [1], [2]. However, primary classrooms are heterogeneous: learners vary in prior vocabulary, acquisition pace, and error patterns. Under these conditions, one-size-fits-all game progression may under-serve both struggling learners and those who advance faster.

A key research and development gap lies in **how adaptivity is engineered and reported**. While learner modeling is central to personalized learning [3], many educational games either (i) remain non-adaptive or (ii) describe adaptivity at a high level without a reproducible specification of mastery updates, parameterization, decision thresholds, and verification artifacts. Even when Bayesian Knowledge Tracing (BKT) is adopted, implementation details that determine validity—such as initialization, slip/guess settings, and update logic are often under-documented, limiting replication and technical auditability in real-time game environments [4], [5].

This paper positions itself as an engineering-focused contribution. We specify and implement a real-time BKT loop inside a Unity-based Android vocabulary game and provide verification evidence that runtime behavior matches the stated model. By consolidating algorithmic specification, parameter configuration, and testable decision policies, the work supports reproducible deployment and provides a foundation for future comparative studies.

Main contributions

1. A reproducible Unity/Android architecture for real-time BKT-driven adaptivity.
2. A single-source-of-truth BKT specification (parameters, initialization, update equations, mastery rule).

p-ISSN :1858-3911 **,** ***e-ISSN : 2614-5405***
https://journal.fkom.uniku.ac.id/ilkom

3. Verification artifacts (functional tests and logic/path-level checks) supporting auditability and replication.
4. An implementation blueprint extensible to controlled effectiveness studies and learning analytics.

2. RESEARCH METHOD

2.1. Research Design

This study employs a design-and-verification approach to develop and technically validate an Android educational game artifact. The primary outcome is a reproducible real-time adaptivity pipeline from learner response capture to Bayesian Knowledge Tracing (BKT) updates and adaptive content routing rather than a causal effectiveness claim. The development process follows the Game Development Life Cycle (GDLC) model [6], [7], [8].

2.2. Elicitation and Learning-Content Preparation

Requirements were elicited via:

1. **Classroom observation** to identify common Grade III learner difficulties in basic English vocabulary and interaction patterns relevant to gameplay (e.g., response behavior, attention span).
2. **Semi-structured teacher interview** to validate instructional needs, determine target vocabulary scope, and agree on feasible practice formats.
3. **Literature review** to ground the adaptivity mechanism in established learner modeling and to ensure methodological consistency [3], [5], [9], [10].

A vocabulary item bank was prepared and organized into thematic sets and difficulty tiers. Each item is assigned: (a) skill/topic label, (b) difficulty tier, and (c) correct answer key. Teacher input is used to validate age appropriateness and content coverage.

2.3. System Development Using GDLC

The system is developed using GDLC phases: **initiation, pre-production, production, testing, and release** [6], [7], [8].

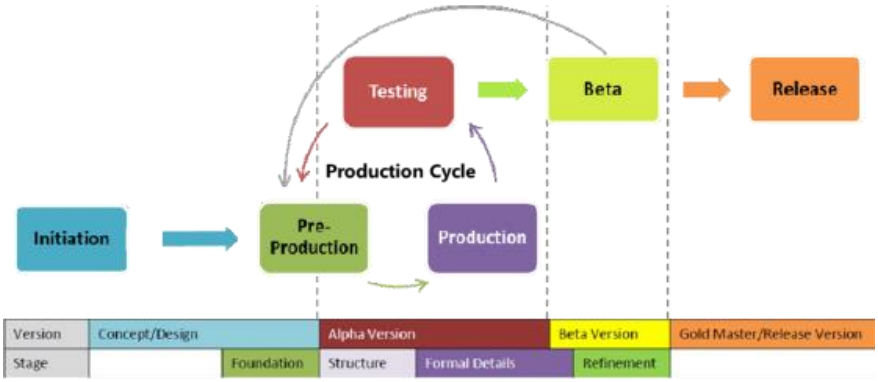


Figure 1. *Game Development Life Cycle Guidelines[6]*

1. **Initiation:** define functional requirements (login/session, practice loop, scoring/feedback, logging) and constraints (Android performance, offline capability).
2. **Pre-production:** produce Game Design Document (GDD), UI/scene flow, and the adaptivity policy (threshold and progression rules).
3. **Production:** implement the game in Unity (C#), integrate assets, implement item-bank manager, logging module, and BKT service/component.
4. **Testing:** execute functional verification and structural verification (Section 2.6).
5. **Release:** package Android build for limited acceptance use focused on runtime stability and deployment feasibility.

2.4. System Architecture Overview

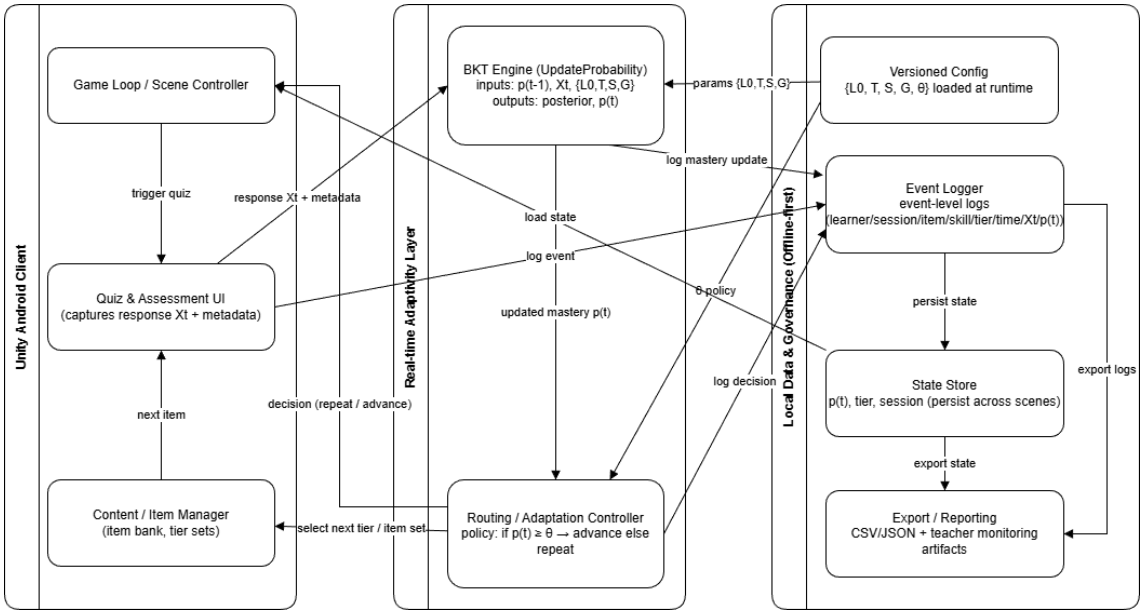


Figure 2. High-Level System Architecture of the BKT-Based Adaptive Game System

Figure 2 summarizes the runtime adaptivity pipeline. The Unity game loop triggers quiz events: learner responses are processed by the BKT engine to update mastery; the routing controller applies the threshold policy to select repetition or progression; and event logs plus adaptive state are stored locally and exported for reporting and reproducibility.

2.5. Bayesian Knowledge Tracing (BKT)

The adaptive engine implements standard BKT to estimate learner mastery probabilistically per skill/topic [3], [5], [9], [10]. For each learner–skill pair, BKT uses four parameters: initial mastery $P(L_0)$, transition/learning $P(T)$, slip $P(S)$, and guess $P(G)$.

Let $p_{t-1} = P(L_{t-1})$be prior mastery and $X_t \in \{0,1\}$be the observed response (1=correct, 0=incorrect). Posterior mastery is computed by Bayes' rule:

If $X_t = 1$(correct):

$$P(L_{t-1} \mid X_t = 1) = \frac{p_{t-1}(1-S)}{p_{t-1}(1-S) + (1-p_{t-1})G}$$

If $X_t = 0$(incorrect):

$$P(L_{t-1} \mid X_t = 0) = \frac{p_{t-1}S}{p_{t-1}S + (1-p_{t-1})(1-G)}$$

Then learning transition is applied:

$$p_t = P(L_{t-1} \mid X_t) + (1 - P(L_{t-1} \mid X_t)) \cdot T$$

Table 1. BKT Parameters

Parameter	Meaning	Impact on updates
(P(L_0))	Early probability of mastering a skill	Determine start mastery
(T)	Probability of learning after one chance	Control mastery ascension
(S)	Slip (wrong despite mastery)	Lowering posterior when wrong
(G)	Guess (true even if you haven't mastered it)	Posterior holding when correct

2.6. Real-Time Adaptation Policy and Parameter Governance

Parameter governance (anti inkonsistensi naskah–kode): all BKT parameters and mastery threshold $\theta$are stored in a **single configuration source** (table/file) loaded at runtime (not hard-coded). Configuration changes are versioned, allowing the paper to reference one definitive set.

Adaptation rule: the game applies a threshold policy: when $p_t \geq \theta$, the learner is routed to a higher difficulty tier or next content segment; otherwise, additional practice is delivered within the current tier. To prevent oscillation, an optional **window rule** can be applied (e.g., threshold must be met for a minimum number of recent items).

If calibration is required, pilot interaction traces can be used to estimate parameters using transparent tooling such as **pyBKT** while retaining clear reporting of the final parameter set [11].

p-ISSN :1858-3911 **,** ***e-ISSN : 2614-5405***
https://journal.fkom.uniku.ac.id/ilkom

2.7. Data Logging and Verification Strategy

Event-level logging is implemented for auditability and replication. Each interaction stores pseudonymous learner/session ID, item ID, skill/topic ID, timestamp, correctness, current mastery p_t, and routing decision.

Table 2. Log Schema for Reproducibility

Field	Description
learner_id (pseudonymous)	Anonymous identity of players
session_id	Play session identity
item_id	Question/item ID
skill_id/topic_id	Tracked skill labels
difficulty_tier	The difficulty level when the item is awarded
timestamp	Interaction time
response (0/1)	False/true
prior_mastery (p_{t-1})	Mastery before update
posterior_mastery	Bayes results before transition
updated_mastery (p_t)	Mastery after transition
routing_decision	Up/fixed/next segment

2.8. Verification and Validation

Verification and validation prioritize evidence for the paper's architecture + verification claim:

(a) Black-box functional verification: test cases validate end-to-end flows (scene transitions, item rendering, answer submission, logging, BKT update invocation, threshold checking, routing).

(b) White-box basis-path verification: the function UpdateProbability() and routing controller are tested using basis-path coverage guided by cyclomatic complexity to ensure each independent decision path is executed [12].

3. RESEARCH RESULT

3.1. Game Flow and Storyboard (Functional Scenes)

The developed artifact implements a structured learning loop consisting of **content exposure → adaptive assessment → mastery-gated progression**. The loop begins from the main menu, continues to vocabulary material access, activates adaptivity via a pretest, and then executes repeated quiz-driven updates during staged gameplay. Table 3 summarizes the storyboard at the level required for an engineering paper focusing on scene purpose and outputs that are relevant to adaptivity and verification.

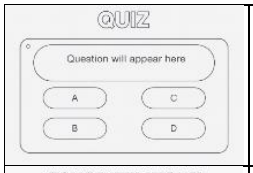
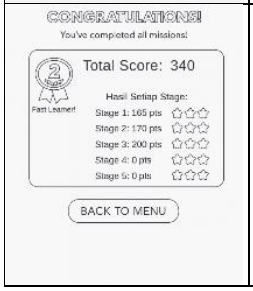
Table 3. Storyboard and adaptivity-relevant outputs

Scene	Core function	Output relevant to adaptivity / verification
Main Menu	Start and navigation	Entry to learning pipeline; controlled access to gameplay modules
Material	Vocabulary exposure	Content gate prior to adaptive activation; ensures exposure before assessment
Pretest	Initialize learner state	Establish initial mastery/level; triggers adaptive mode and stores state
Gameplay (Adventure stage)	Engagement loop	Context for assessment triggers; stage completion depends on routing decision
Quiz	Assessment + mastery update	Per-item BKT update; produces updated mastery used for routing
Achievement	Feedback	Displays progression outcome consistent with routing logic

Table 4. Storyboard Game Design

Scene	Visual	Action
	Play, Setting, Info	Start Game
	Vocabulary material	Can't be skipped without interaction
	Moving characters	Avoiding enemies & items
	Initial level	Defining initial capabilities

p-ISSN :1858-3911 **,** ***e-ISSN : 2614-5405***
https://journal.fkom.uniku.ac.id/ilkom

	Adaptive	BKT regulates difficulties
	Badge	Player achievements

3.2. User Interface Implementation

User interfaces are implemented for two roles: **student-facing gameplay** and **teacher-facing monitoring**. For students, the implemented scenes include the main menu, system information, login, vocabulary material, platformer gameplay, quiz interaction, and achievement summary (Figures 3–9). For teachers, the system provides restricted access to monitoring and reporting functions, enabling result viewing and report export to support classroom deployment and traceability (Figure 9).

- **Figure 3–9** present the student-facing workflow (menu → material → gameplay → quiz → achievement).
- **Figure 10** shows the teacher dashboard interface for monitoring/reporting.



Figure 3. Main Menu View



Figure 4. System Information Display

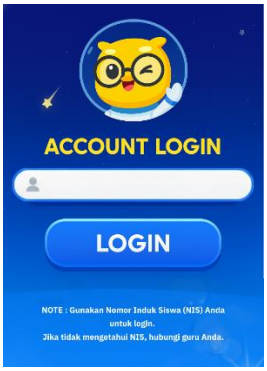


Figure 5. Game Login Menu View

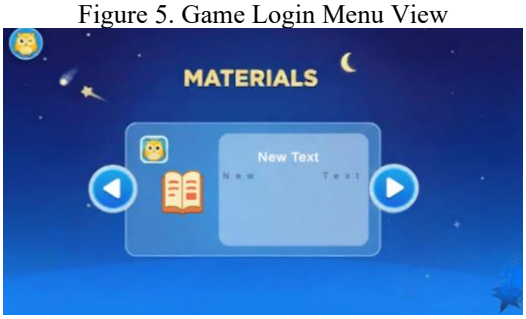


Figure 6. Material Menu View

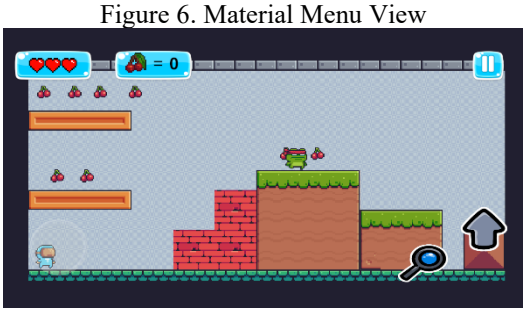

Figure 7. Playing Display



Figure 8. Quiz Menu View



Figure 9. Achivement Menu View

p-ISSN :1858-3911 **,** ***e-ISSN : 2614-5405***
https://journal.fkom.uniku.ac.id/ilkom

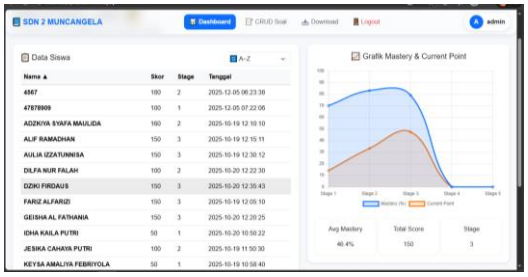


Figure 10. Teacher Dashboard Interface

3.3. Real-Time BKT Adaptivity Implementation

Real-time adaptivity is implemented at the granularity of each quiz interaction. The runtime loop is executed as follows:

1. The system presents a quiz item aligned with the current stage/difficulty tier.
2. The learner submits a binary response $X_t \in \{0,1\}$(correct/incorrect).
3. The BKT engine computes posterior mastery conditioned on $X_t$and applies the learning transition to obtain the updated mastery probability $P(L_t)$(as defined in Section 2.4).
4. The adaptation router compares $P(L_t)$against a mastery threshold $\theta$and selects the next action: **repeat within current tier** or **advance to a higher tier/next stage**.

At the implementation level, the mastery variable is stored as an internal state (e.g., pKnown), and the learning/transition parameter is represented as pLearn, corresponding to $P(L)$and $P(T)$in the BKT formulation. The transition update follows:

$$pKnown \leftarrow posterior + (1 - posterior) \cdot pLearn$$

This design enables immediate mastery updating and routing decisions during runtime, supporting **real-time adaptivity** rather than post-hoc personalization.

Reproducibility requirement (single-source-of-truth parameters). To support independent audit and replication, all parameters $\{P(L_0), P(T), P(S), P(G), \theta\}$are defined in a versioned configuration source loaded at runtime (not hard-coded). The evaluated build is therefore traceable to one definitive parameter set, reducing the risk of manuscript–implementation divergence.

3.4. System Testing Results (Black-Box Functional Verification)

Black-box functional verification was conducted to confirm correct end-to-end behavior across navigation, content gating, adaptive-mode activation, persistence, and mastery-gated progression. While the complete test suite covers broader usability and flow control, Table 5 reports the subset of cases that directly evidence the correctness of the adaptive pipeline: adaptive initiation, response-level mastery updating, persistence of adaptivity state, and deterministic routing based on θ.

Table 5. Adaptivity-critical black-box verification (selected cases)

Case ID	Scenario verified	Trigger / test case	Expected observable output	Result
BB-03	Adaptive mode activation	In Material, confirm readiness	Transition to Pretest and adaptive mode active	Pass
BB-05	Mastery update responsiveness	Answer a pretest/ quiz item correctly	Mastery increases relative to baseline	Pass
BB-06	Persistence of adaptivity decision	Complete pretest	Difficulty/ level state is stored (persisted)	Pass
BB-08	Failure routing below threshold	End stage with mastery (< \theta)	“Retry” state/panel appears	Pass
BB-09	Success routing at/above threshold	End stage with mastery (\ge \theta)	“Success” state/panel appears	Pass
BB-11	Progression continuity	Continue after success	Transition to next stage	Pass

hese outcomes indicate that the adaptive mechanism is not only implemented, but also produces stable, observable, and persistent routing decisions aligned with mastery conditions.

3.5 White-Box Verification Results (Basis-Path for UpdateProbability())

White-box verification targets the BKT update routine (UpdateProbability()), as it is the algorithmic nucleus of adaptivity. Structurally, the routine contains a correctness-dependent decision branch (correct vs incorrect) followed by posterior computation and a learning

p-ISSN :1858-3911 **,** ***e-ISSN : 2614-5405***
https://journal.fkom.uniku.ac.id/ilkom

transition update. To validate internal control-flow coverage, basis-path testing was applied. Cyclomatic complexity indicates two independent paths corresponding to the correctness branch; therefore, adequacy is achieved by executing test inputs that force both branches to execute at least once.

Table 6. Basis-path coverage for UpdateProbability()

Path ID	Input condition	Independent path	Branch covered	Expected outcome (test oracle)
WB-P1	correct = true	Start → correct-branch → posterior → transition → return	Correct-answer posterior	Posterior uses ((1-S)); mastery updated via transition; returns updated mastery
WB-P2	correct = false	Start → wrong-branch → posterior → transition → return	Incorrect-answer posterior	Posterior uses (S); mastery updated via transition; returns updated mastery

his evidence strengthens auditability by demonstrating that the internal decision structure of the mastery update routine is explicitly covered, not merely assumed correct from user-level behavior.

Overall, the results demonstrate a deployable Unity/Android educational game artifact with a functioning real-time BKT adaptivity loop. Adaptivity is evidenced by (i) per-response mastery updating, (ii) mastery-threshold routing that produces deterministic retry/success outcomes, (iii) persistence of adaptive state across scenes, and (iv) verification artifacts at both functional (black-box) and structural (basis-path) levels. These results support the paper's engineering contribution: a reproducible architecture with testable adaptivity behavior suitable for technical audit and replication.

4 DISCUSSION

4.1 Principal Findings and Their Meaning

This study demonstrates that Bayesian Knowledge Tracing (BKT) can be embedded as a **real-time control mechanism** inside a Unity-based Android educational game. Mastery is updated after each quiz interaction and immediately used to govern routing decisions. Unlike many adaptive learning reports that describe personalization at a conceptual level, this work operationalizes adaptivity as an explicit runtime pipeline—response capture → mastery update → threshold decision → next-item selection—consistent with the role of knowledge tracing as a learner-state estimator for practice sequencing [3], [5], [9].

The contribution is intentionally framed as **technical validity and reproducibility**, not as an effective claim. Within this scope, the results indicate that the adaptive pipeline is (i) implemented end-to-end, (ii) observable at the system level (e.g., deterministic retry/success routing), and (iii) traceable through persisted learner state and reporting/export mechanisms.

4.2 Why This Work Matters: Reproducible Adaptivity in a Mobile Game Context

BKT is widely established, and recent syntheses underline its continued relevance while also noting recurring issues in reporting and comparability across implementations [10]. Many adaptive systems remain difficult to reproduce because manuscripts often omit critical implementation details: the exact parameter set, initialization procedure, mastery threshold logic, and verification artifacts needed for technical audit.

This paper positions its novelty in **implementation transparency** by treating the adaptivity mechanism as a reproducible system component rather than an implicit feature. This is particularly relevant in mobile game environments, where algorithmic behavior must remain stable under scene transitions, local persistence constraints, and practical runtime limitations.

4.3 Strength of Evidence: Complementary Verification Layers

The verification strategy directly supports credibility. **Black-box functional verification** confirms that the adaptive flow behaves as intended from the user perspective: adaptive mode activation, persistence of level decisions, and mastery-gated routing outcomes are observable and consistent. **White-box basis-path verification** strengthens this evidence by demonstrating that the internal decision structure

p-ISSN :1858-3911* , *e-ISSN : 2614-5405
https://journal.fkom.uniku.ac.id/ilkom

of the mastery update routine is structurally exercised across independent execution paths.
Using cyclomatic complexity and independent-path execution as an adequacy argument aligns with established software testing principles [12]. Together, these layers reduce the risk that adaptivity is under-specified, difficult to audit, or inconsistent between narrative and runtime behavior.

4.4 Parameter Governance: A Non-Negotiable Condition for Reproducibility

A key lesson from the implementation is that **parameter consistency is inseparable from reproducibility**. In BKT, values for slip $P(S)$, guess $P(G)$, transition/learning $P(T)$, and the mastery threshold θdirectly shape posterior updates and therefore routing behavior [3], [5], [9]. Even small deviations can materially alter mastery trajectories and progression decisions.
For publication-grade reporting, the system should enforce a **single-source-of-truth parameter specification**: a versioned configuration (table/file) loaded at runtime and referenced explicitly in the manuscript. This ensures that the parameter set described in the paper matches the evaluated build. When parameters deviate from defaults (e.g., $P(G) = 1/k$for multiple-choice items), the deviation should be justified either through domain rationale or calibration using interaction traces. In this context, using established estimation tooling (e.g., pyBKT) provides a transparent pathway for parameter fitting and reporting [11].

4.5 Limitations and Threats to Validity

Several limitations should be acknowledged to avoid over-claiming:

1. **Scope limitation (no causal learning claim).** The study validates system correctness and reproducibility; it does not establish that adaptivity improves learning outcomes relative to a non-adaptive baseline.
2. **Model assumptions.** Classic BKT uses a binary observation model and typically assumes a single latent mastery per skill without explicitly modeling forgetting or item difficulty heterogeneity. These assumptions can be challenged in vocabulary learning where item difficulty may vary substantially [3], [10].
3. **Parameter sensitivity and threshold fragility.** Fixed parameterization and a fixed mastery threshold can yield brittle routing if item quality varies or if learners adopt strategic guessing. Without calibration or sensitivity analysis, robustness cannot be inferred.
4. **Verification boundaries.** Basis-path coverage is demonstrated for the mastery update kernel, but other modules (routing controller, persistence, export/sync) may also affect reproducibility. Functional verification supports end-to-end behavior; however, deeper structural testing of routing and persistence logic would further strengthen auditability.

4.6 Future Work

This study establishes a technical baseline that enables a structured research program:

1. **Parameter calibration and reporting:** estimate $P(L_0)$, $P(T)$, $P(S)$, and $P(G)$per skill/tier from logged traces and publish the final parameter table with procedure details [11].
2. **Sensitivity analysis:** evaluate how routing outcomes change across plausible θranges and parameter perturbations to demonstrate robustness.
3. **Controlled comparative evaluation:** implement a non-adaptive baseline (fixed progression) and compare against BKT-driven adaptivity to quantify learning effects while controlling exposure time and novelty effects.
4. **Reproducibility package:** publish parameter configuration, pseudocode aligned with the evaluated build, and compact test suites (functional and structural) as supplementary material to support independent replication.

5 CONCLUSION

This paper presented a design-and-verification study that implements **real-time Bayesian Knowledge Tracing (BKT)** within a Unity-based Android educational game for vocabulary practice. The proposed system operationalizes adaptivity as a runtime pipeline in which each learner response triggers immediate mastery updating and the updated mastery probability governs deterministic routing decisions for practice continuation or progression.
The work contributes: (1) a deployable Unity/Android architecture integrating response capture, BKT updating, routing policy, and event logging; (2) an auditable adaptivity mechanism supported by persisted learner state and exportable reporting; and (3) verification evidence at two complementary levels—black-box functional validation of the adaptive flow and white-box basis-path verification of the

p-ISSN :1858-3911* , *e-ISSN : 2614-5405
https://journal.fkom.uniku.ac.id/ilkom

UpdateProbability() routine—supporting technical correctness and reproducibility.
This study is intentionally scoped to technical validity and reproducibility rather than causal learning effectiveness. Future work should focus on (i) enforcing a single-source-of-truth parameter configuration aligned with the evaluated build, (ii) parameter calibration and sensitivity analysis to demonstrate routing robustness, and (iii) controlled comparative studies (adaptive vs non-adaptive) to evaluate whether real-time BKT adaptivity yields measurable learning gains beyond engagement effects.

6 SUGGESTIONS

1. Standardize and version all BKT parameters $\{P(L_0), P(T), P(S), P(G), \theta\}$in a single configuration source aligned with the evaluated build to ensure reproducibility and prevent manuscript–implementation inconsistency.
2. Strengthen verification coverage by adding unit tests for routing control and persistence/export modules, complementing the existing basis-path testing of UpdateProbability().
3. Calibrate and stress-test the adaptive policy through pilot-log parameter calibration and sensitivity analysis on $\theta$to confirm routing robustness under realistic learner behaviors.
4. Extend future evaluation with controlled comparison (adaptive vs non-adaptive) and richer learning indicators (e.g., retention and time-on-task) to assess effectiveness beyond technical validity.

REFERENCES

[1] P. Wouters, C. van Nimwegen, H. van Oostendorp, and E. D. van der Spek, "A meta-analysis of the cognitive and motivational effects of serious games.," *J. Educ. Psychol.*, vol. 105, no. 2, pp. 249–265, May 2013, doi: 10.1037/a0031311.

[2] D. H. Dixon, T. Dixon, and E. Jordan, "Second language (L2) gains through digital game-based language learning (DGBLL): A meta-analysis," *Language Learning & Technology*, vol. 26, no. 1, pp. 1–25, Feb. 2022, doi: 10.64152/10125/73464.

[3] R. Pelánek, "Bayesian knowledge tracing, logistic models, and beyond: an overview of learner modeling techniques," *User Model. User-adapt. Interact.*, vol. 27, no. 3–5, pp. 313–350, Dec. 2017, doi: 10.1007/s11257-017-9193-2.

[4] G.-H. Gweon, H.-S. Lee, C. Dorsey, R. Tinker, W. Finzer, and D. Damelin, "Tracking student progress in a game-like learning environment with a Monte Carlo Bayesian knowledge tracing model," in *Proceedings of the Fifth International Conference on Learning Analytics And Knowledge*, New York, NY, USA: ACM, Mar. 2015, pp. 166–170. doi: 10.1145/2723576.2723608.

[5] A. T. Corbett and J. R. Anderson, "Knowledge tracing: Modeling the acquisition of procedural knowledge," *User Modelling and User-Adapted Interaction*, vol. 4, no. 4, pp. 253–278, 1995, doi: 10.1007/BF01099821.

[6] R. Andriyat Krisdiawan, "PENERAPAN MODEL PENGEMBANGAN GAMEGDLC (GAME DEVELOPMENT LIFE CYCLE)DALAM MEMBANGUN GAME PLATFORM BERBASIS MOBILE," vol. 2, no. 1, 2019.

[7] R. Ramadan and Y. Widyani, "Game development life cycle guidelines," in *2013 International Conference on Advanced Computer Science and Information Systems (ICACSIS)*, IEEE, Sep. 2013, pp. 95–100. doi: 10.1109/ICACSIS.2013.6761558.

[8] S. Aleem, L. F. Capretz, and F. Ahmed, "Game development software engineering process life cycle: a systematic review," *Journal of Software Engineering Research and Development*, vol. 4, no. 1, p. 6, Dec. 2016, doi: 10.1186/s40411-016-0032-7.

[9] A. T. Corbett and J. R. Anderson, "Knowledge tracing: Modeling the acquisition of procedural knowledge," *User Modelling and User-Adapted Interaction*, vol. 4, no. 4, pp. 253–278, 1995, doi: 10.1007/BF01099821.

[10] I. Šarić-Grgić, A. Grubišić, and A. Gašpar, "Twenty-five years of Bayesian knowledge tracing: a systematic review," *User Model. User-adapt. Interact.*, vol. 34, no. 4, pp. 1127–1173,

p-ISSN :1858-3911 **,** ***e-ISSN : 2614-5405***
https://journal.fkom.uniku.ac.id/ilkom

Sep. 2024, doi: 10.1007/s11257-023-09389-4.

[11] O. Bulut, J. Shin, S. N. Yildirim-Erbasli, G. Gorgun, and Z. A. Pardos, "An Introduction to Bayesian Knowledge Tracing with pyBKT," *Psych*, vol. 5, no. 3, pp. 770–786, Jul. 2023, doi: 10.3390/psych5030050.

[12] T. J. McCabe, "A Complexity Measure," *IEEE Transactions on Software Engineering*, vol. SE-2, no. 4, pp. 308–320, Dec. 1976, doi: 10.1109/TSE.1976.233837.