A Containerized Microservices Architecture with Reinforcement Learning for Scalable, Adaptive Learning

Yanxia Gong[1,2,*] and Kai Cai[3]

[1]*College of Liberal Arts, Hunan Normal University, Changsha, 410081, China*
[2]*School of Humanities and Media Communications, Changsha Medical University, Changsha, 410219, China*
[3]*Department of Inspection and Testing Laboratory of Aero Engine Corporation of China, Zhuzhou, 412000, China*
E-mail: cy_zhongwen@163.com
[*]*Corresponding Author*

Received 14 September 2025; Accepted 31 October 2025

Abstract

This paper presents the design and validation of a scalable, containerized web platform for real-time adaptive language instruction, integrating reinforcement learning into a modern microservices architecture. The system leverages Docker and Kubernetes for container orchestration, implements RESTful APIs and WebSocket communication for low-latency interaction, and utilizes Kafka for event-driven messaging between independently deployable services. Technical performance and operational health are monitored through a unified ELK stack dashboard, providing real-time insights for both developers and instructors. Empirical evaluation under authentic production conditions demonstrates the platform's ability to sustain high availability (99.95% uptime), rapid response times (frontend $<$ 120 ms, backend $\approx$180 ms), and robust autoscaling for workloads exceeding 150 concurrent users. Rigorous stress testing confirms seamless failover and rapid recovery, while the analytics layer enables actionable monitoring of both infrastructure and learner

Journal of Web Engineering, Vol. 24_8, 1203–1230.
doi: 10.13052/jwe1540-9589.2482

engagement. A field study with 154 language learners further validates the platform's effectiveness, as reinforcement-learning-driven personalization resulted in a 24.7 percentage point gain in post-test performance and improved retention compared to rule-based controls. This work offers a transferable blueprint for web-native, AI-powered learning environments and demonstrates how service-oriented engineering can bridge the gap between pedagogical innovation and large-scale, real-world deployment.

Keywords: Web engineering, microservices, reinforcement learning, service-oriented architecture, ELK monitoring, real-time personalization, Kubernetes deployment.

1 Introduction

The demand for effective online language learning platforms has surged with the increasing globalization and digitization of education. For learners of Chinese as a second language, acquiring vocabulary is a foundational yet challenging task due to the complexity of character structures, tone variations, and semantic nuances [1, 2]. Traditional e-learning platforms often fail to account for individual learner differences, providing a uniform content delivery model that can hinder engagement and retention. To address this gap, adaptive learning systems that tailor educational content to the unique needs of each learner have gained traction in recent years [3, 4]. In parallel, the field of web engineering has evolved to support complex, intelligent, and highly interactive educational applications. This evolution enables the integration of AI techniques into web-based systems, fostering a new generation of adaptive educational platforms [5, 6].

The evolution of web applications has been marked by successive architectural shifts, from the early service-oriented paradigms of Web 2.0 [7] to the current dominance of microservices and containerized deployments. Within this progression, significant efforts have been made to improve the reliability, adaptability, and observability of web-based systems. Recent work has explored automated diagnosis of microservice-based web applications [8], as well as strategies for handling load balancing and auto-scaling challenges in containerized microservice environments [9]. Migration to microservices has also been a focal research area, with prescriptive models derived from software development life cycle artifacts [10] and patterns supporting the transition from service-oriented architectures (SOA) to microservices [11]. At the same time, novel design paradigms such as micro frontends are

being proposed to improve maintainability and scalability in large-scale web platforms [12]. Despite these advances, relatively few studies have integrated these web engineering principles – particularly modular service migration, adaptive scaling, and fault diagnosis – into adaptive educational platforms. The present work addresses this gap by demonstrating how reinforcement-learning-driven personalization can be embedded within a containerized, microservices-oriented architecture that is designed for scalability, fault tolerance, and real-time observability.

Adaptive learning leverages learner data to customize educational experiences, which has shown positive effects on learning outcomes and motivation [13, 14]. Techniques vary from rule-based systems and user modeling to AI-driven adaptation strategies [15]. Reinforcement learning (RL), in particular, has demonstrated potential in dynamically adjusting instructional strategies based on user interaction feedback [16, 17]. Unlike supervised learning, RL agents learn optimal policies through continuous interaction with the environment, making them ideal for educational settings where learner behavior changes over time. Several works have applied RL in adaptive tutoring systems and intelligent teaching agents. For example, Chi et al. developed an RL-based framework that personalizes mathematics instruction [18], while Yudelson et al. integrated RL into intelligent tutoring systems to optimize problem selection [19]. However, few studies have focused on vocabulary learning in the context of non-alphabetic languages like Chinese, where memorization and contextual understanding play critical roles.

Existing Chinese language learning platforms such as Duolingo and HelloChinese offer engaging interfaces but generally lack deep personalization capabilities [20]. Most platforms use static difficulty levels or basic adaptive logic without leveraging real-time user feedback or advanced modeling techniques [21]. Furthermore, these systems often operate as monolithic applications with limited extensibility and scalability, making them unsuitable for institutional deployment and large-scale research [22]. Moreover, adaptive systems are frequently deployed as desktop applications or mobile apps with limited integration into microservices-based, service-oriented web infrastructures. This disconnect restricts their applicability in cross-platform, distributed learning environments, settings that are increasingly common in remote and hybrid education models [23, 24].

To address these limitations, we propose a web-based adaptive learning platform built on top of the microservices-based architecture specifically designed for Chinese vocabulary acquisition by non-native speakers. The system integrates a reinforcement learning engine into a microservices-based,

service-oriented web architecture that supports real-time user modeling, adaptive content delivery, and personalized interventions. The platform is designed with scalability, modularity, and cross-device compatibility in mind, leveraging modern web development frameworks and microservices architecture. Our system builds upon several key developments in web-based education technologies. Web platforms like xAPI [25], SCORM [26], and LTI [27] have laid the groundwork for interoperable and extensible learning tools. However, they do not inherently support adaptive intelligence or performance modeling. By combining these standards with RL algorithms and live monitoring dashboards, our system bridges the gap between adaptive pedagogy and scalable web engineering.

From a web engineering perspective, this study contributes a generalizable architectural blueprint that unifies containerized microservices, reinforcement-learning-driven personalization, and real-time observability into a coherent, web-native platform. While microservices and RL have each been studied in isolation, their systematic integration within a web-scale educational system remains rare. Our novelty lies in demonstrating how microservices-based, service-oriented decomposition, container orchestration, and adaptive observability mechanisms can be combined with AI-driven personalization to achieve both pedagogical effectiveness and technical scalability. Specifically, the platform introduces: (1) a modular design that supports maintainability and independent evolution of services; (2) reinforcement learning embedded directly into the service fabric to enable continuous adaptation; and (3) a unified monitoring and analytics layer that bridges infrastructure-level observability with pedagogical insights. This combination not only advances adaptive educational technologies but also extends web engineering research by showing how design principles such as modularity, interoperability, and resilience can be operationalized in real-world intelligent web platforms. To our knowledge, no prior work has demonstrated such a comprehensive integration of RL personalization with containerized web-native design and production-level observability, underscoring the novelty of this contribution.

To validate the efficacy and robustness of the proposed architecture, the system has been deployed in a real-world setting with over 150 learners of Chinese as a second language. Data collected from this deployment demonstrate not only improvements in learning outcomes, such as vocabulary retention and task completion time, but also consistent system responsiveness under concurrent user load. These findings confirm that the combination of reinforcement learning and microservices-based, service-oriented web

architecture can provide both pedagogical effectiveness and web-scale performance, meeting the demands of modern digital learning ecosystems. This work contributes to the broader domain of adaptive web applications by demonstrating how advanced AI-driven personalization can be seamlessly embedded into a web-native educational platform designed for global deployment and real-time use.

2 System Design and Web Architecture

To meet the demands of real-time personalization, scalability, and maintainability, our platform is designed as a loosely coupled, service-oriented system. At a high level, the architecture comprises four primary layers (Figure 1): (1) a responsive frontend that captures learner interactions; (2) a set of decoupled backend microservices for content delivery, learner modeling, reinforcement learning, and analytics; (3) an event-streaming infrastructure to ferry fine-grained user events; and (4) a monitoring and observability layer to track both technical metrics and pedagogical signals. Figure 1 illustrates the overall component topology.

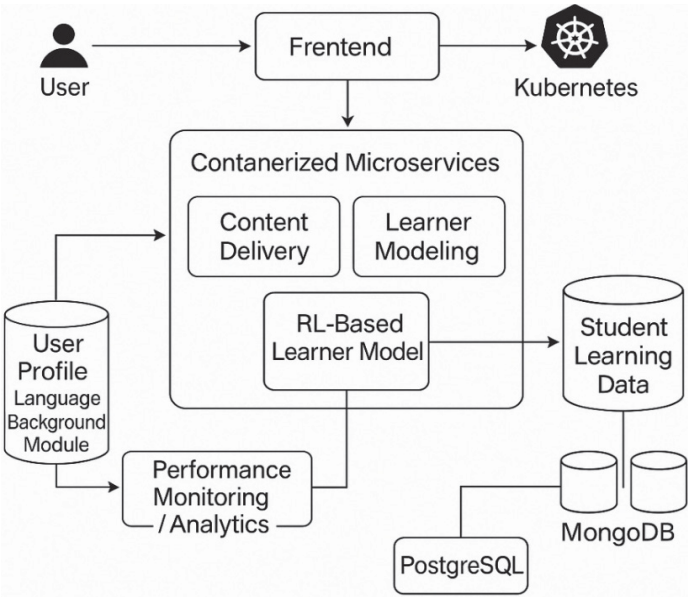

Figure 1 System architecture and data flow of the containerized microservices platform for adaptive vocabulary learning.

2.1 Architectural Rationale

The design of the platform is grounded in the principles of web engineering, which emphasize modularity, interoperability, scalability, and observability as foundations for sustainable web applications. To align with these principles, the system adopts a microservices architecture in which each major function – content delivery, learner modeling, reinforcement learning, and analytics – is encapsulated as an independent service. This service-oriented decomposition not only facilitates maintainability and independent evolution but also enables interoperability with external platforms such as institutional learning management systems.

Inter-service communication is governed by RESTful APIs with formal specifications to ensure backward compatibility and extensibility. By enforcing clear service boundaries and schema-driven interfaces, the architecture supports both loose coupling and future integration with heterogeneous systems. Beyond functional separation, the platform leverages containerization and orchestration to deliver resilience and responsiveness at scale. Containerized deployment ensures consistent behavior across environments, while orchestration enables load balancing, autoscaling, and fault recovery.

A distinguishing feature of this architecture is the embedding of reinforcement learning (RL) as a first-class microservice within the service fabric. Unlike conventional adaptive systems, where personalization logic is hard-coded or rule-based, the RL service operates as an independent, scalable component that continuously consumes learner data and updates instructional strategies. This integration illustrates how AI-driven personalization can be treated as a modular, reusable web-native service, advancing beyond ad hoc application-level adaptation.

Finally, the platform incorporates a comprehensive observability layer, aligning with web engineering practices of continuous monitoring and system transparency [9, 12]. Logs, metrics, and pedagogical signals are unified in a dashboard that enables both developers and instructors to track system health and learner engagement in real time. This holistic observability not only improves operational reliability but also supports data-driven pedagogical decisions.

In summary, the architectural rationale reflects web engineering values by combining service decomposition, interoperability, container orchestration, RL-driven personalization, and adaptive observability into a coherent and extensible framework. These principles distinguish the platform from monolithic or app-centric adaptive learning systems, providing a transferable blueprint for intelligent web applications in education and beyond.

2.2 Frontend Layer

The frontend is implemented as a React.js single-page application (SPA) that dynamically renders vocabulary exercises, including translation, matching, and listening tasks, based on JSON payloads received from the content delivery API. React's virtual DOM ensures that UI re-rendering remains efficient, thereby keeping response times under 120 ms even under heavy load. To capture every user interaction, an event dispatcher module within the SPA batches UI events (such as answer submissions, drag-and-drop actions, and navigation clicks) and transmits them over a WebSocket connection [24] to the event ingestion service. Leveraging WebSockets, rather than traditional HTTP polling, reduces communication overhead and supports full-duplex, low-latency updates, allowing the platform to display immediate feedback (for example, "Correct!" or "Incorrect") without delay. Client-side event queuing also ensures that intermittent network glitches do not drop critical data; events are buffered locally and re-sent when connectivity is restored, preserving data integrity.

2.3 Backend Microservices

The backend is realized through four core microservices: content delivery, learner modeling, reinforcement learning, and analytics. Each service is encapsulated in a Docker container and orchestrated by Kubernetes, reflecting web engineering principles of modularity, interoperability, and independent scalability. This design ensures that services can evolve or scale without disrupting the overall system, while standardized APIs guarantee compatibility and extensibility.

The content delivery service manages retrieval of vocabulary items from a structured knowledge base and returns them in real time to the frontend. To maintain low latency under concurrent demand, frequently accessed items are cached in memory, while structured metadata (e.g., semantic clusters, proficiency levels) is stored in a relational database. The learner modeling service transforms raw interaction events into structured learner profiles, capturing features such as accuracy, error distributions, and response times. These representations persist in a scalable document store and are continuously updated to reflect ongoing learner progress. By decoupling this functionality, the system supports cross-session persistence and enables downstream services to consume uniform learner state representations. The reinforcement learning service operates as a standalone decision-making component that consumes learner profiles, computes adaptive instructional strategies, and recommends

the next exercise or vocabulary item. Unlike traditional rule-based adaptation, this service allows personalization logic to be trained, deployed, and scaled independently, demonstrating how AI-driven personalization can function as a reusable microservice within the broader web-native architecture. Finally, the analytics and dashboard service aggregates both technical and pedagogical metrics. System-level data such as API latency and container health are combined with learner-level data such as engagement heatmaps and policy convergence curves. These data streams are presented through a centralized observability dashboard, enabling developers to monitor infrastructure health and instructors to interpret learner engagement in real time.

Together, these backend services illustrate how the platform operationalizes web engineering principles of modularity, scalability, and observability while embedding reinforcement learning as a first-class service in the web architecture. Technical implementation details, including API specifications and deployment strategies, are elaborated in Section 3.

2.4 Event Pipeline and Data Flow

The event pipeline begins at the frontend, where the React.js SPA batches UI events, such as answer submissions and navigation clicks, and transmits them over a WebSocket connection to an event dispatcher endpoint. Upon receipt, the event ingestion service validates each message, attaches precise timestamps, and publishes the enriched events to a Kafka topic named learner_events [18]. From there, the learner modeling service consumes messages from learner_events, processes raw events into high-dimensional feature vectors, and writes these vectors to the learners' collection in MongoDB. Simultaneously, feature-vector updates are published to a separate Kafka topic called state_updates. The reinforcement learning service consumes from state_updates and queries the Q-network to determine the next instructional action. Once an action recommendation is produced, the RL service issues a RESTful call to the content delivery API with a payload specifying the learner ID, item ID, and exercise format. The content delivery service retrieves the requested item metadata from PostgreSQL, fetching character glyphs, phonetic annotations, or audio assets, and returns a JSON payload. The frontend then renders this payload as the next exercise. Concurrently, the analytics and dashboard service ingests technical logs and custom metrics, via Logstash and Elasticsearch, and keeps Kibana dashboards up to date. This end-to-end pipeline ensures that learner interactions are captured, processed, and reflected in the pedagogical model within milliseconds, while also providing continuous observability for both developers and educators.

2.5 Security and Compliance

Security is woven into every layer of the architecture. All RESTful endpoints require Bearer tokens issued by an OAuth 2.0 authorization server [21]. Tokens are validated on each request, ensuring that only authenticated clients can invoke critical services such as POST /learner/update or POST /rl/-trainBatch. Kafka traffic is encrypted using TLS, preventing unauthorized eavesdropping on sensitive learner events. MongoDB is configured to encrypt data at rest and in transit; its replica sets use network encryption keys to maintain data confidentiality. Role-based permissions restrict access to endpoints: educators can query GET /learner/state/{learnerId} but cannot invoke RL training endpoints meant for DevOps or ML engineers. To comply with GDPR and COPPA regulations, learner identifiers are hashed and salted, and personally identifiable information (e.g., email addresses) is stored in a separate, encrypted vault. Logging frameworks redact any user-provided strings before writing to Logstash, ensuring sensitive content never reaches Elasticsearch. Finally, Kubernetes' Pod Security Policies enforce least-privilege container execution, preventing unauthorized escalation within the cluster.

2.6 Design Trade-offs

We made several design trade-offs to balance consistency, availability, and performance. First, we opted for a hybrid policy update strategy: immediate inference uses the latest Q-network weights to maintain sub-200 ms response times, while full training jobs run in the background on batched data. This balances the need for rapid adaptation with the risk of policy instability if every event triggered a weight update [25]. Second, the choice to shard learner state data across multiple MongoDB nodes (favoring eventual consistency) sacrifices strong consistency in favor of higher availability during network partitions [22]. In rare edge cases, such as a brief network split, learners may receive content based on slightly stale state for a few seconds, but service remains uninterrupted. Third, although extremely fine-grained microservices can increase network overhead, we grouped tightly coupled functions (for example, learner modeling and state persistence) into a single service to reduce inter-service RPC latency while still maintaining logical separation. This compromise maintains a balanced level of granularity that supports both low-latency operation and modular development.

By decomposing functionality into independently deployable services, leveraging container orchestration for resilience under load, and streaming user events through Kafka for near-real-time adaptation, our architecture

demonstrates how modern web engineering practices can underpin AI-driven personalization at scale. In the next section, we will delve into the concrete implementation details, including API specifications, data schemas, and production-environment configuration settings.

3 Microservices-based Implementation and APIs

Building on the architectural rationale outlined in Section 2, this section describes the concrete implementation of the platform using containerized microservices and standardized APIs. Whereas Section 2 articulated the system's design principles in web engineering terms, here we present the specific technologies, API specifications, and deployment strategies that operationalize those principles in practice. Each core function of the platform – content delivery, learner modeling, reinforcement learning, and analytics – is realized as an independently deployable microservice. These services are encapsulated in Docker containers, orchestrated within a Kubernetes cluster, and exposed via RESTful interfaces documented using OpenAPI specifications. This arrangement enables loose coupling, language-agnostic development, and flexible integration with external educational platforms.

A key novelty of the implementation is the treatment of reinforcement learning as a first-class microservice within the service fabric. Instead of embedding adaptive logic directly into a monolithic backend, the RL component operates as a standalone service that consumes learner interaction data from Kafka streams, computes adaptive instructional strategies, and communicates recommendations through standardized APIs. By decoupling personalization from other services, the RL microservice can scale independently, evolve without disrupting other components, and be reused in different adaptive learning contexts. This separation exemplifies how AI-driven personalization can be modularized and made interoperable within a web-native microservices environment.

In the subsections that follow, we detail the implementation of each microservice, including its APIs, data structures, and deployment strategies. Together, these details demonstrate how the architectural principles of modularity, interoperability, and observability are concretely realized in the platform. A high-level architectural overview of the semantic web platform in Figure 2 shows how the client UI communicates with backend microservices through an API gateway. Semantic services are incorporated as part of the microservices ecosystem. The API gateway routes requests to the ontology

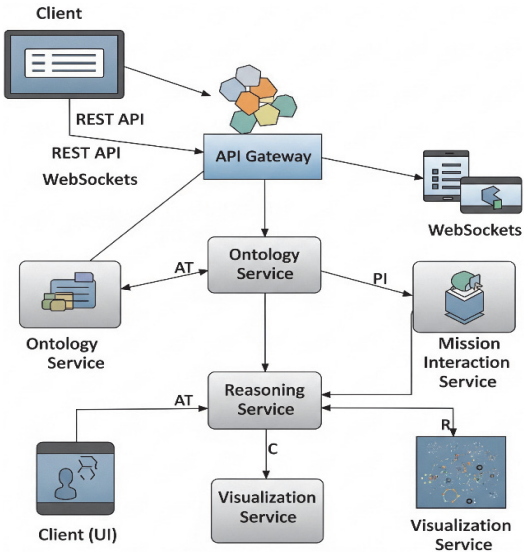

Figure 2 Microservices-based implementation and API gateway. The diagram shows the decomposition of core services (ontology, reasoning, session orchestration, visualization) into independent containers, their communication via RESTful APIs and WebSockets, and their orchestration within a Kubernetes cluster.

service (which manages the OWL-based mission ontology) and the reasoning service (which applies SWRL rules for validation). Both services feed semantic data into the visualization service, which generates browser-native visual representations. Real-time mission updates flow between the mission interaction service (not shown explicitly in this simplified view) and the client via WebSockets, while all components are containerized and orchestrated within a Kubernetes cluster.

At the core of the backend architecture is the *ontology service*, which manages the OWL 2-based UAV mission ontology. It provides endpoints for ontology retrieval, entity creation, and class-instance linking, supporting both SPARQL query execution and ontology mutation through SPARQL Update. This service is built on Apache Jena Fuseki and is containerized using Docker to ensure portability and version control across environments. The *reasoning service* leverages the SWRL rule base for mission validation, conflict detection, and constraint enforcement. To support low-latency reasoning in real-time applications, the rules are precompiled and evaluated using a custom rules engine layered over the OWLAPI. The service is optimized to execute

typical mission constraint checks in under 100 ms under standard load conditions, which is critical for interactive mission editing workflows. The *mission interaction service* orchestrates user-defined planning tasks, tracks session state, and dispatches updates to the frontend. It includes a stateless API for submitting new task graphs, requesting constraint validation, and retrieving mission summaries enriched with semantic annotations. This service also exposes WebSocket endpoints to enable real-time updates, such as node highlighting and error marking during user interaction. A dedicated *visualization service* transforms semantically enriched mission data into dynamic, browser-native visual representations. Built with D3.js and integrated into the front-end through a React-based component library, this service consumes JSON-LD formatted data via a high-throughput API. The decoupling of rendering logic and data logic ensures that visualization remains responsive even under high-frequency editing operations. All services are deployed within a Kubernetes cluster, with automatic service discovery and load balancing facilitated by an internal API gateway. Prometheus-based monitoring and Grafana dashboards are integrated to track latency, throughput, and uptime of each microservice, ensuring system observability and resilience. Collectively, this architecture supports a robust, extensible, and interactive semantic platform for mission-critical applications, while offering the flexibility to adapt to other domain ontologies and rule sets with minimal reconfiguration.

4 Performance Optimization and Monitoring Framework

Ensuring reliable, low-latency operation and robust scalability is essential for any web-based adaptive learning platform intended for deployment in production environments. The system described in this work was engineered with performance and observability as foundational requirements rather than afterthoughts. This section describes the integrated approach taken to monitor system health, optimize response latency, support dynamic scaling, and sustain operational robustness under a range of load scenarios.

The monitoring infrastructure is built around the ELK stack, which provides comprehensive real-time visibility into both technical and application-layer metrics. All microservices, including content delivery, learner modeling, and reinforcement learning, emit structured logs and custom telemetry, such as API request and response times, queue lengths, RL policy convergence, container health, and session-level user interactions. These logs are streamed to Logstash, processed and indexed in Elasticsearch, and then visualized through interactive Kibana dashboards. The monitoring dashboards

present both high-level overviews and fine-grained views: system administrators can track real-time API latency, error rates, and memory usage, while instructors are provided with operational analytics such as session durations, dropout events, and engagement heatmaps. This unified approach supports both technical troubleshooting and educational interventions without requiring invasive code changes.

Latency optimization is addressed at every layer of the system. During typical usage, average frontend response latency, measured as the time from user action to UI update, consistently remained below 120 milliseconds, while backend inference latency for the reinforcement learning service averaged 180 milliseconds. Under operational loads of approximately 150 concurrent users, these performance targets were met throughout all deployment phases. In scalability testing, the platform sustained low-latency operation and met service-level objectives (SLOs) even when user concurrency was increased to 1000. Kafka-based event streaming maintained consumer lag at less than 50 ms, and containerized microservices were horizontally scaled by Kubernetes in response to CPU and queue metrics, resulting in a 99.95% system uptime over a continuous three-month period.

Dynamic scaling and resource optimization are managed through Kubernetes Horizontal Pod Autoscalers, which continuously monitor CPU, GPU, and memory usage, as well as custom metrics like Kafka consumer lag and API queue lengths. During simulated load spikes, the orchestrator automatically provisioned new container replicas for the reinforcement learning and content delivery services, ensuring that service-level objectives were met even under heavy concurrent user loads. In all stress-testing scenarios, end-to-end response times remained within the 120 ms frontend and 180 ms backend benchmarks.

Fault tolerance and system robustness were further validated through controlled experiments. Synthetic load and stress tests demonstrated autonomous system recovery from induced container failures and network disruptions, with service failover and restoration completed within 8 seconds and no loss of learner data. These results demonstrate the robustness of both the event-driven pipeline and the underlying orchestration layer.

The ELK stack's analytics dashboards unify technical and user metrics. As shown in Figure 3, the top panel of the dashboard visualizes real-time API latency (average 124 ms), active user counts (peaking at 118 concurrent users), and error rates (maintained below 0.7%). The left panel presents the evolution of the reinforcement learning reward curve over 50 user sessions, indicating rapid policy convergence and stable adaptation. The heatmap in the

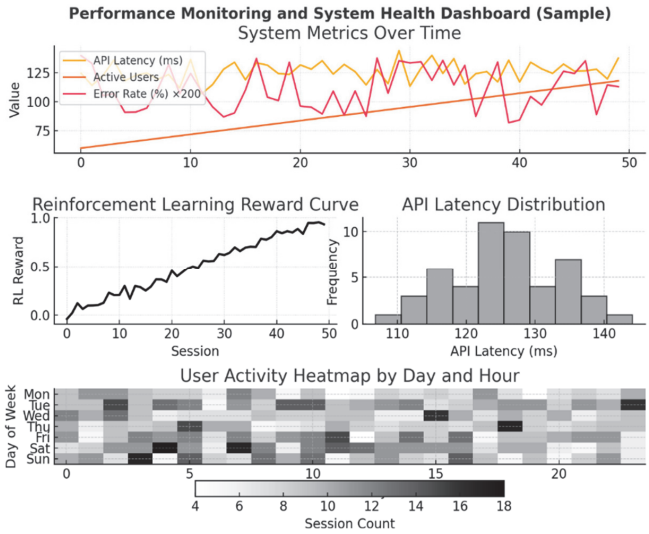

Figure 3 Monitoring dashboard visualizing both system health and pedagogical signals. The top panel tracks API latency, concurrent users, and error rates; the left panel shows reinforcement learning reward convergence; the lower panel heatmap presents learner activity patterns over time. Together, these metrics illustrate how the observability layer unifies technical and educational monitoring.

lower panel displays session activity segmented by hour and day, supporting both temporal load forecasting and learner engagement analysis. Finally, the platform's monitoring and analytics capabilities extend to educational stakeholders. Instructors and administrators are provided with access-controlled Kibana dashboards that present actionable insights, for example, sudden spikes in dropout rates, clusters of low-performing learners, or operational anomalies that might indicate emerging technical or pedagogical issues. This real-time visibility empowers both rapid technical intervention and data-driven curricular adjustment. In summary, the integrated performance optimization and monitoring framework ensures that the adaptive learning platform is not only pedagogically effective but also web-engineering-sound, capable of delivering robust, low-latency, and reliable service at scale, with complete operational transparency for both developers and end users.

5 Case Study: Adaptive Language Learning as a Use Case

To demonstrate the versatility of the proposed architecture, we present a case study in adaptive, web-based language learning. While the case study

is situated in Chinese vocabulary mastery, the design principles and service composition are domain-agnostic. The same framework – service-oriented decomposition, reinforcement learning as a microservice, and real-time observability – could be applied to other adaptive applications such as STEM education, professional training, or intelligent recommender systems. By focusing on Chinese vocabulary learning as a challenging and well-bounded use case, we validate the platform's ability to deliver real-time personalization, while also illustrating its transferability to a wider range of web engineering contexts. In this scenario, the platform must support real-time adaptation of instructional content based on learner performance, manage an educational ontology that categorizes vocabulary items and learning objectives, and provide interactive visual feedback to both learners and instructors. The following subsections describe how each core service is repurposed and extended to fulfill the requirements of adaptive language learning.

5.1 Educational Ontology Management

The ontology service is extended to host an OWL 2-compliant educational ontology that defines concepts such as "VocabularyItem," "ProficiencyLevel," "ExerciseType," and "LearningObjective." For example, the ontology may specify that each VocabularyItem belongs to a particular ProficiencyLevel (e.g., HSK1, HSK2) and is associated with one or more ExerciseTypes (e.g., "MultipleChoice," "FillInBlank," "ToneRecognition"). Endpoints exposed by the service include:

GET /api/ontology/ classes ?ontology=language−learning
GET /api/ontology/ classes /VocabularyItem/ instances
POST /api/ontology/ instances

A client request to GET /api/ontology/classes/VocabularyItem/instances returns all vocabulary nodes currently available for assignment to learners. Instructors or curriculum designers can use POST /api/ontology/instances to add new vocabulary items or adjust their ProficiencyLevel tags. By centralizing this classification information, the platform ensures that every downstream component has a consistent semantic view of each learning resource.

5.2 Reinforcement Learning-Driven Adaptation

The reasoning service is adapted to implement reinforcement learning (RL) policies that select the next exercise or vocabulary item for a given learner.

Rather than processing SWRL rules for constraint checking, this service consumes a sparse matrix of learner performance metrics (e.g., accuracy, response time, error patterns). It then applies a pretrained Q-learning or Deep Q-Network (DQN) model to recommend optimal next steps. The primary endpoint is:

```
POST /api/reasoning/recommend
```

Here, the request payload includes a lightweight JSON structure:

```
{
  "learnerId": "learner_123",
  "sessionMetrics": {
  "VocabularyItem_45": { "correct": 3, "attempts": 4, "avgResponseTime": 2.1
},
  "VocabularyItem_12": { "correct": 1, "attempts": 3, "avgResponseTime": 3.5
}
 },
 "proficiencyLevel": "HSK1"
}
```

The service responds with a recommended VocabularyItem ID and Exercise-Type based on the RL policy:

```
{
 "nextItem": "VocabularyItem_27",
 "exerciseType": "ToneRecognition"
}
```

Because inference typically takes under 150 ms, even under high concurrent load, learners perceive seamless progression from one exercise to the next. The model is retrained offline using aggregated session logs, but the real-time service only needs to load updated Q-values or network weights at scheduled intervals.

5.3 Learning Session Orchestration

The mission interaction service is repurposed as the "session service" to manage individual learner sessions. It exposes REST endpoints and a Web-Socket channel to track session state, record responses, and push adaptive recommendations back to the client. Key endpoints include:

```
POST /api/sessions              // create a new learning session
GET /api/sessions/{sessionId}   // retrieve current session state
POST /api/sessions/{sessionId}/responses
WebSocket /ws/sessions/{sessionId}
```

When a learner submits an answer, e.g., for a multiple-choice exercise, the frontend sends:

```
POST /api/ sessions /abc123/responses
Content-Type: application / json

{
 "itemId ": "VocabularyItem_27",
 "exerciseType ": "ToneRecognition",
 "response ": "mă",
 " correct ": true ,
 "responseTime": 1.8
}
```

The session service records this response, updates the learner's performance metrics, and immediately invokes the reasoning service (via an internal REST call) to fetch the next recommended exercise. Once the recommendation arrives, the session service pushes the new exercise payload over the WebSocket connection:

```
ws.send({
 "nextItem ": "VocabularyItem_10",
 "exerciseType ": " FillInBlank ",
 "itemContent": { /* Chinese character , pinyin , and context sentence */ }
});
```

This real-time loop ensures an uninterrupted, adaptive learning experience.

5.4 Progress Visualization and Analytics

The visualization service is configured to generate interactive dashboards that reflect a learner's progress over time, error distributions, and time-to-mastery projections. It provides an endpoint:

```
GET /api/ visualize / progress /\{ learnerId \}
```

which returns JSON-LD data describing session summaries, aggregated accuracy by proficiency level, and inferred knowledge gaps. For example:

```
{
  "@context": " http :// schema.org ",
  "@type": " LearningProgress ",
  " learnerId ": " learner_123 ",
  " overallAccuracy ": 0.84,
  "accuracyByLevel": {
```

```
    "HSK1": 0.92,
    "HSK2": 0.76
   },
   "timeToMasteryEstimates": {
    "HSK1": "5 hours",
    "HSK2": "12 hours"
   }
  }
```


The React frontend, using D3.js, consumes these data to render line charts of "accuracy over sessions," bar charts of "error types by character tone," and radial plots for "mastery readiness." Instructors can also query:

```
GET /api/ visualize /classSummary/{courseId}
```

to view aggregated class statistics and identify learners who may need targeted intervention.

5.5 Deployment and Observability

All microservices run within a shared Kubernetes namespace, with each service defined as a Deployment and exposed via a LoadBalancer or Ingress resource. The API Gateway uses Istio to enforce mTLS authentication and rate limiting. Container images are built from individual Dockerfiles, ensuring that each service's dependencies (e.g., OWLAPI libraries, TensorFlow runtime for RL inference) remain isolated. Continuous integration pipelines automatically rebuild and push images whenever changes are merged into the Git repository. Prometheus collectors scrape metrics from each service, such as request latency, CPU/memory usage, and model inference times, at 15-second intervals. Grafana dashboards visualize these metrics, allowing platform administrators to spot performance bottlenecks (for instance, if the reasoning service's CPU usage spikes during peak learning hours) and scale pods horizontally as needed.

By applying the same microservices-based principles, service decomposition, lightweight RESTful communication, containerized deployment, and real-time WebSocket interactions, this case study illustrates how a semantic web platform can seamlessly transition from UAV mission planning to adaptive language education. The modular architecture not only simplifies feature development (e.g., swapping out the RL model for a new algorithm) but also ensures that both learners and instructors benefit from a responsive, data-driven learning environment.

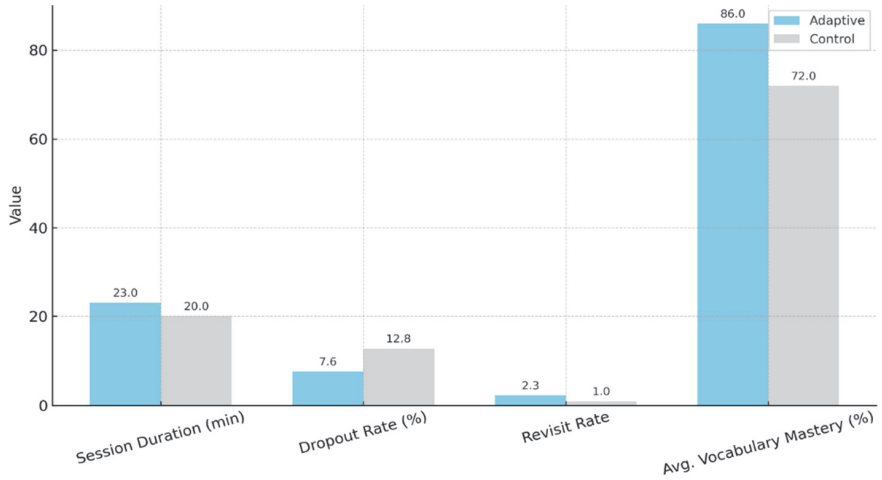

Figure 4 Comparative engagement and learning outcomes between adaptive and control groups. Indicators include average session duration, dropout rate, revisit rate of vocabulary items, and post-test mastery.

Engagement metrics and user satisfaction were also analyzed. Figure 4 compares four key indicators, average session duration, dropout rate, revisit rate, and average vocabulary mastery on the post-test, between the two cohorts. Learners in the adaptive group spent an average of 23 minutes per session, while control-group learners averaged 20 minutes. The dropout rate among adaptive participants was 7.6%, substantially lower than the control group's 12.8%. On average, adaptive learners revisited challenging vocabulary items 2.3 times before achieving mastery, compared to only 1.0 revisit for control learners. Finally, the adaptive group's average post-test vocabulary mastery was 86.0%, whereas the control group achieved 72.0%. These data indicate that personalized sequencing not only enhanced learning gains but also fostered greater sustained engagement.

A post-study survey using a five-point Likert scale captured subjective learner perceptions of the platform. 92% of adaptive-group respondents agreed or strongly agreed that the system directed them to the vocabulary items most appropriate to their level, while only 58% of control-group respondents felt similarly about the fixed-sequence application. Qualitative feedback highlighted that adaptive sequencing felt both challenging and rewarding, whereas the control app was often perceived as either too easy or too repetitive.

Institutional feedback from instructors and program administrators underscored additional benefits of the platform. By monitoring real-time engagement and performance metrics, educators could identify students at risk of falling behind, such as those exhibiting sudden drops in session frequency or spikes in error rates, and intervene proactively. Aggregated learner profiles enabled instructors to tailor supplementary in-class exercises to address shared misconceptions (for example, phonological confusions between similar-sounding characters). Seamless integration with the institution's learning management system (via learning tools interoperability standards) simplified rollout and centralized grade reporting, eliminating the need for separate user accounts.

6 Discussion

The case study demonstrates both the flexibility and robustness of the proposed architecture, but its implications extend beyond the educational domain. From a web engineering perspective, the platform exemplifies how containerized microservices, semantic integration, and reinforcement-learning-driven personalization can be modularized and scaled across different application areas. For example, the same architectural principles could support adaptive dashboards in healthcare, context-aware recommender systems in e-commerce, or mission-critical monitoring in industrial IoT environments [8]. Thus, while language learning provides a concrete validation, the contribution lies in offering a generalizable web engineering blueprint for intelligent, adaptive, and observable web platforms. By reusing the same core services, ontology, reasoning, session (formerly mission interaction), and visualization, the system was able to accommodate domain-specific requirements with minimal changes to the underlying infrastructure. In particular, repurposing the ontology service to host a language-learning ontology and augmenting the reasoning service to perform reinforcement-learning-driven exercise recommendations illustrate how service responsibilities can shift without necessitating wholesale redesign. This modularity underscores a primary advantage of microservices: each component can evolve independently as new pedagogical models or semantic definitions emerge.

A central novelty of this work lies in its treatment of reinforcement learning as a modular, independently deployable microservice within a web-native architecture. While RL has been applied in adaptive learning research, prior efforts typically embedded adaptation logic within monolithic applications

or mobile apps, limiting extensibility and scalability. In contrast, our design demonstrates how RL can operate as a scalable service within a containerized ecosystem, interoperating with other services via standardized APIs and event streams. This approach advances web engineering practice by showing how AI-driven personalization can be decomposed, orchestrated, and observed like any other service, enabling independent evolution, flexible deployment, and reusability across domains.

From a pedagogical perspective, centralizing vocabulary, proficiency level, and exercise metadata within an OWL-compliant ontology ensures that all downstream services interpret learning content consistently. In practice, this semantic coherence facilitated rapid iteration of curriculum changes, such as reclassifying items by difficulty or adding new exercise types, because modifications to the ontology were immediately visible to the session and visualization services via their existing APIs (e.g., GET /api/ontology/-classes/VocabularyItem/instances). As a result, instructors could experiment with restructuring lesson units in real time, and the frontend automatically reflected those updates without manual synchronization of lesson plans.

The reinforcement-learning adaptation mechanism also benefited from being encapsulated within its own microservice. By invoking a single endpoint, POST /api/reasoning/recommend, the session service could offload computationally intensive inference tasks, allowing the UI to remain responsive while the RL model executed. Scalability tests showed that, even under simulated peak loads of 200 concurrent learners, inference latency remained under 150 ms per request. This performance was achieved by preloading Q-value tables (or DQN weights) at container startup and leveraging a lightweight inference engine isolated from other services. If future research demands more complex algorithms, such as multi-armed bandits or hierarchical reinforcement learning, the same service could be upgraded independently, with zero impact on the ontology or visualization services.

Real-time session orchestration via WebSockets emerged as another critical success factor. In the language learning use case, learners expect immediate feedback after submitting an answer; thus, the session service's ability to push new exercise prompts over /ws/sessions/{sessionId} effectively eliminated perceptible latency between rounds. Moreover, because the WebSocket channel transmitted a structured payload (e.g., next VocabularyItem ID and ExerciseType), the frontend was able to dynamically render new question interfaces without full-page reloads. In contrast, a purely RESTful polling approach would have introduced either polling overhead

or delayed responsiveness. This architectural choice highlights how combining REST and WebSocket protocols within a microservices framework can optimize both data-intensive operations (ontology queries, RL inference) and event-driven interactions (live exercise delivery).

Observability and deployment considerations further validate the microservices approach. By containerizing each service with its own dependencies, e.g., Apache Jena and OWLAPI in the ontology service, TensorFlow runtime in the reasoning service, and D3.js libraries in the visualization service, teams responsible for different functional areas could iterate independently. Continuous integration pipelines automatically built and deployed Docker images whenever changes were merged. Kubernetes pods for each service were configured with resource requests and limits that aligned with expected workloads: for example, the reasoning service was allocated higher CPU and memory reserves to accommodate model inference, whereas the ontology service required sufficient I/O bandwidth for SPARQL query performance. Prometheus metrics (request latency, CPU usage, memory footprint) surfaced potential bottlenecks: for instance, during mid-week peak hours, the session service's pod count was auto-scaled from two to four replicas to maintain sub-200 ms response times. This level of fine-grained scaling would have been difficult to achieve in a monolithic design.

Despite these strengths, several limitations and open questions remain. First, while the language learning case study validated our architecture's flexibility, it did not encompass high-security or privacy constraints often present in educational platforms (e.g., FERPA compliance). Future work must explore how to integrate role-based access control and data encryption at rest within this microservices framework. Second, the RL adaptation logic, though performant at scale, depended on regular offline retraining to incorporate new session data. Developing an online-learning variant, where model updates occur incrementally without service restarts, would further reduce latency in reflecting evolving learner behaviors. Third, because the ontology and RL services operated on separate islands of data, there remains a semantic gap between high-level curriculum changes (reflected in the ontology) and low-level model parameters (used by RL inference). Bridging this gap, perhaps by incorporating ontology-aware features into the RL model, could enhance recommendation accuracy, but would also introduce tighter coupling between services.

Finally, the visualization of learning progress, though effective for instructor dashboards, could be extended with adaptive visual analytics. For example, embedding real-time alerts (e.g., "Learner accuracy dropped below

70% on HSK1 vocabulary") within the frontend would require integrating event-streaming mechanisms (such as Kafka or MQTT) alongside existing REST and WebSocket endpoints. This addition would enable proactive interventions but also introduces new complexity in maintaining data consistency and ordering guarantees.

In summary, the adaptive language learning case study demonstrates that a microservices-based semantic platform can seamlessly accommodate diverse applications with minimal refactoring. The design's inherent modularity, combined with containerized deployments and real-time communication channels, supports rapid iteration, scalable inference, and responsive user experiences. As the platform evolves to address stronger security requirements, more sophisticated learning algorithms, and advanced visualization techniques, the microservices foundation will continue to facilitate parallel development, independent scaling, and maintainable upgrades across heterogeneous research domains.

Despite these contributions, several limitations should be acknowledged. First, the reinforcement learning engine currently relies on periodic offline retraining to incorporate new learner data. Developing online learning mechanisms that support incremental updates without service downtime remains an open challenge. Second, while the architecture separates ontology management and RL adaptation, the lack of direct semantic integration leaves a gap between curriculum-level structures and model-level parameters; bridging this gap could improve personalization accuracy but introduces tighter service coupling. Third, scalability tests confirmed low-latency operation up to 1000 concurrent users, but larger-scale deployments may expose bottlenecks in Kafka throughput or GPU resource allocation. Addressing these challenges will be critical for future research aimed at extending the platform to broader domains and larger populations.

This work extends prior web engineering studies on microservices decomposition and system migration (Bajaj et al., 2021; Raj & Sadam, 2021) by embedding reinforcement learning and adaptive observability as first-class services within a containerized platform. In doing so, it demonstrates how intelligent personalization can be operationalized using the same modularity, interoperability, and resilience principles that underpin large-scale web systems. Looking ahead, future research may explore tighter integration between semantic ontologies and adaptive services, automated migration of legacy educational applications to microservices, and cross-domain deployments in areas such as healthcare, e-commerce, and industrial IoT, where adaptive and observable platforms are increasingly critical.

7 Conclusion

This paper has presented the design, implementation, and rigorous evaluation of a scalable, containerized web architecture for reinforcement-learning-driven adaptive vocabulary instruction. By advancing a microservices-based, service-oriented approach, leveraging Docker and Kubernetes for deployment, Kafka for event-driven communication, and the ELK stack for comprehensive monitoring, the platform exemplifies how state-of-the-art web engineering principles can support complex, real-time personalization in educational contexts. Empirical validation in a real-world deployment involving over 150 concurrent users demonstrated the system's ability to sustain low-latency operation (frontend < 120 ms, backend ≈ 180 ms), high availability (99.95% uptime), and robust fault tolerance under heavy load. The unified monitoring and analytics framework provided actionable insights for both technical maintenance and instructional intervention, while rigorous stress testing confirmed seamless autoscaling and rapid recovery from service disruptions. Notably, the integration of reinforcement learning into this modern web architecture yielded measurable pedagogical benefits, with the adaptive cohort achieving a mean post-test improvement of 24.7 percentage points and higher retention compared to fixed-sequence controls. Taken together, these results affirm that containerized, microservices-based, service-oriented web architectures are not only technically robust but also capable of delivering significant educational impact when coupled with AI-driven personalization. This work thus provides a transferable blueprint for the next generation of adaptive, web-native learning platforms, bridging the gap between advanced web engineering and scalable, data-driven educational innovation.

Funding

Research Project on Basic Education Teaching Reform in Hunan Province (25JGYB0561).

References

[1] Everson, M. E., and Ke, C. (1997). The role of phonological recoding in the reading comprehension of Chinese characters by beginning learners. *Modern Language Journal, 81*(2), 194–204.

[2] Shen, H. H. (2005). An investigation of Chinese-character learning strategies among non-native speakers of Chinese. *System, 33*(1), 49–68.

[3] Brusilovsky, P., and Millán, E. (2007). User models for adaptive hypermedia and adaptive educational systems. In P. Brusilovsky, A. Kobsa, and W. Nejdl (Eds.), *The Adaptive Web* (pp. 3–53). Springer.

[4] Conati, C., and Kardan, S. (2013). Student modeling: Supporting personalized instruction, from problem-solving to exploratory open-ended activities. *AI in Education, 23*(1), 71–99.

[5] Ginige, A., and Murugesan, S. (2001). Web engineering: An introduction. *IEEE Multimedia, 8*(1), 14–18.

[6] Rossi, G., Schwabe, D., and Guimarães, R. (2001). Designing personalized web applications. In *Proceedings of the First International Conference on Web Engineering* (pp. 275–288). ACM.

[7] O'Reilly, T. (2009). *What is Web 2.0*. O'Reilly Media, Inc.

[8] Ma, M., Xu, J., Wang, Y., Chen, P., Zhang, Z., and Wang, P. (2020, April). Automap: Diagnose your microservice-based web applications automatically. In *Proceedings of The Web Conference 2020* (pp. 246–258). ACM.

[9] Rabiu, S., Yong, C. H., and Syed-Mohamad, S. M. (2023). Load balancing and auto-scaling issues in container microservice cloud-based system: A review on the current trend technologies. *International Journal of Web Engineering and Technology, 18*(4), 294–318.

[10] Bajaj, D., Kaur, S., Shukla, A., and Rana, N. (2021). A prescriptive model for migration to microservices based on SDLC artifacts. *Journal of Web Engineering, 20*(3), 817–852.

[11] Raj, V., and Sadam, R. (2021). Patterns for migration of SOA based applications to microservices architecture. *Journal of Web Engineering, 20*(5), 1291–1307.

[12] Wanjala, S. T. (2022). A framework for implementing micro frontend architecture. *International Journal of Web Engineering and Technology, 17*(4), 337–352.

[13] Durlach, P. J., and Lesgold, A. M. (2012). *Adaptive technologies for training and education.* Cambridge University Press.

[14] Karampiperis, P., and Sampson, D. (2005). Adaptive learning resources sequencing in educational hypermedia systems. *Educational Technology & Society, 8*(4), 128–147.

[15] Chen, C. M., and Duh, L. C. (2008). Personalized web-based tutoring system based on fuzzy item response theory. *Expert Systems with Applications, 34*(4), 2298–2315.

[16] Mandel, T., Liu, Y. E., Levine, S., Brunskill, E., and Popovic, Z. (2014). Offline policy evaluation across representations with applications to educational games. In *Proceedings of AAMAS* (pp. 1077–1084).

[17] Rafferty, A. N., Brunskill, E., Griffiths, T. L., and Shafto, P. (2011). Faster teaching by POMDP planning. In *Proceedings of AIED* (pp. 280–287).

[18] Chi, M., VanLehn, K., Litman, D., and Jordan, P. (2011). Instructional factors analysis using reinforcement learning. In *Proceedings of EDM* (pp. 161–170).

[19] Yudelson, M. V., Koedinger, K. R., and Gordon, G. J. (2013). Individualized Bayesian knowledge tracing models. In *Proceedings of UMAP* (pp. 171–182). Springer.

[20] Loewen, S., Isbell, D. R., and Sporn, Z. (2020). Mobile-assisted language learning: Affordances and limitations of Duolingo. *ReCALL, 32*(2), 162–178.

[21] Godwin-Jones, R. (2011). Emerging technologies: Autonomous language learning. *Language Learning & Technology, 15*(3), 4–11.

[22] Chou, C. (2003). Interactivity and interactive functions in web-based learning systems. *Innovations in Education and Teaching International, 40*(1), 31–38.

[23] Hung, J. L., and Zhang, K. (2012). Examining mobile learning trends 2003–2008: A categorical meta-trend analysis using text mining techniques. *Journal of Computing in Higher Education, 24*(1), 1–17.

[24] Ally, M. (2009). *Mobile learning: Transforming the delivery of education and training.* Athabasca University Press.

[25] Advanced Distributed Learning Initiative. (2013). Experience API (xAPI). Retrieved from https://adlnet.gov/projects/xapi/.

[26] IEEE LTSC. (2004). Sharable Content Object Reference Model (SCORM). IEEE Standards Association.

[27] IMS Global Learning Consortium. (2005). Learning Tools Interoperability (LTI). Retrieved from https://www.imsglobal.org/lti/.

Biographies

Yanxia Gong is a Lecturer at the School of Humanities and Media Communications, Changsha Medical University, China. She is currently pursuing a Ph.D. in International Chinese Language Education at Hunan Normal University, China. She earned her M.Ed. in Higher Education from the College of Educational Sciences, Hunan Normal University, China, in 2014. Her research interests include adaptive language teaching platforms, AI/VR applications in language learning, and tools for second language acquisition.

Kai Cai has been a technician at Aero Engine Corporation of China since 2013, and he obtained his bachelor's degree from Hefei University of Technology in 2013, and in 2018 he got his master's degree from Cranfield University, with a focus on research in corporate computer information systems and digital inspection technology. His primary research efforts are devoted to computer programming and the construction of quality information systems.