The object of the study is automated generation and validation of wagon interface schemas in railway ticket booking systems. Manual updating of one configuration takes approximately 24 hours. It depends on software releases, requires engineer involvement, and increases the risk of production errors.

A two-layer agentic architecture is proposed. It includes separation of reasoning and execution, multimodal conversion of schematics into structured configurations, and deterministic validation. The architecture was tested on the Ukrzaliznytsia platform. The platform supports more than 150 wagon configurations and serves over 20 million passengers per year.

Twenty production schematics of different wagon classes were processed. One schematic required manual correction. The others were refined through one or two additional prompts. The time required to create a new wagon type decreased from approximately 24 hours to 15 minutes. This corresponds to an approximately 99% reduction. Monthly engineering involvement was largely replaced by a serverless workflow with infrastructure costs of less than 5 USD per month. No incidents caused by model hallucinations were recorded.

The solution supports more than 150 configurations without loading all schemas into a single context. It enables non-technical administrators to update configurations outside release cycles and reduces dependence on engineering teams. The approach integrates mandatory pre-deployment validation into the configuration creation process. Structure, format, and correctness remain subject to formal control. This reduces manual engineering work without changes to the core application code. The approach is applicable to railway ticketing and related systems where physical objects are represented as digital interface schemas

Keywords: ticketing automation, reasoning-execution separation, production validation, computer vision, serverless deployment

UDC 004.89:656.2
DOI: 10.15587/1729-4061.2026.363741

DEVELOPMENT OF A TWO-LAYER AGENTIC ARCHITECTURE FOR AUTOMATED GENERATION AND VALIDATION OF WAGON USER INTERFACE SCHEMAS USING MULTIMODAL LARGE LANGUAGE MODELS

Ivan Dobrovolskyi
Master of Science in AI Engineering
Independent Researcher
Elan Village Lane, 371, San Jose,
California, USA, 95134
E-mail: dobrovolsky94@gmail.com
ORCID: https://orcid.org/0009-0005-6938-0384

Received 09.03.2026
Received in revised form 20.05.2026
Accepted date 29.05.2026
Published date 30.06.2026

How to Cite: *Dobrovolskyi, I. (2026). Development of a two-layer agentic architecture for automated generation and validation of wagon user interface schemas using multimodal large language models. Eastern-European Journal of Enterprise Technologies, 3 (2 (141)), 75–86. https://doi.org/10.15587/1729-4061.2026.363741*

1. Introduction

Railway transportation systems face a persistent digitalization problem. Physical wagon layouts must be translated into booking interfaces with high precision. Online sales increasingly dominate passenger rail service delivery. In 2024, Ukrzaliznytsia transported more than 20 million passengers, and about 86% bought tickets online [1, 2]. Railway fleets may be highly heterogeneous. A single route may include several wagon types with different seats, amenities, and accessibility features. Any mismatch between physical and digital layouts may cause booking errors, passenger complaints, and operational inefficiency.

In modern conditions, this problem is becoming more important because online railway ticketing is no longer an auxiliary service. It is a primary passenger-facing digital channel through which passengers select trains, wagons, seats, services, and accessibility options. Therefore, the digital representation of a wagon must correspond to the physical layout with high precision.

The relevance of automated wagon interface schema generation is determined by current operational conditions. Railway fleets remain heterogeneous. Therefore, one booking platform must support many wagon types. It must reflect different seat layouts. It must represent amenities separately. It must also provide accurate digital representation of accessibility configurations.

Digital railway services must be updated faster than traditional software release cycles allow. Production booking interfaces also require a high level of safety. Layout errors may affect passenger transactions and operational processes.

In practice, this study can reduce the time needed to create and update wagon configurations. It can also decrease dependence on engineering teams and support faster deployment of new wagon types. In addition, it reduces the risk of incorrect passenger-facing layouts in national railway systems.

Therefore, automated generation and validation of wagon user interface schemas is scientifically relevant. It addresses the need to convert physical wagon layouts into accurate, safe, and release-independent digital representations.

2. Literature review and problem statement

Railway digitalization forms the first group of studies relevant to the problem of automated wagon interface schema generation. In paper [3], railway digitalization is considered as a way to accelerate the modernization of legacy railway systems. However, this work focuses on the general need for

digital transformation in railways and does not address the conversion of wagon layouts into passenger-facing booking interface schemas.

In paper [4], next-generation railway traffic management is analyzed through the OPTIMA project. This study confirms the importance of intelligent digital solutions for railway infrastructure. However, its focus is traffic management rather than the representation of wagon interiors in booking platforms.

In paper [5], digitalization and machine learning are applied to railway operations and maintenance. The study shows that artificial intelligence-based methods can improve operational railway processes. However, the problem of automated configuration of passenger-facing wagon user interfaces remains outside its scope.

Large language models provide the second group of technological foundations. In paper [6], large language models are reviewed as tools for code generation. This confirms their potential for structured output synthesis. However, general code generation differs from wagon schema generation because a wagon schema must preserve seat order, layout constraints, direction attributes, accessibility elements, and rendering compatibility.

Paper [7] confirms the usefulness of code-trained models for code generation, but does not consider validated domain-specific schemas. Paper [8] presents Gemini as a multimodal model for processing text, images, and code, but multimodal input does not guarantee schema validity. Paper [9] demonstrates that Gemini 1.5 can process long multimodal contexts. This is useful for technical documentation and large configuration sets. However, long-context capacity does not ensure bounded editing. The model must modify only the relevant schema fragment and preserve the rest of the configuration.

Paper [10] uses LLMs to extract parameters and detect inconsistencies in clinical documentation. The study confirms the need for validation, as false positives and domain-specific errors may occur. This domain differs from railway booking interfaces, where invalid schemas can affect passenger transactions. Paper [11] describes an LLM-based tool for vulnerability detection. The study confirms the importance of adaptation, validation, and reliability control. However, it does not address validation of AI-generated interface schemas before deployment in a production environment.

Study on digital twins provides a third group of relevant sources. Paper [12] considers AI-based digital twins of trains as a means of improving railway services. However, this study focuses primarily on the train as a digital object for operational and predictive purposes, while passenger-facing booking interface schemas remain outside its scope.

Paper [13] proposes a reference architecture for AI-based digital twins in smart railways. This work demonstrates how artificial intelligence can be integrated into railway digital infrastructure. However, it does not provide a workflow for transforming manufacturer wagon schematics into validated interface configurations.

Paper [14] analyzes the integration of artificial intelligence and digital twins into railway maintenance. The study identifies data integration and model accuracy as complex technical issues. These issues relate to wagon schema generation, but the work does not address booking systems or pre-production validation of interface schemas.

Paper [15] examines change management in passenger transport on Ukrainian railways under wartime conditions. This confirms the practical importance of adaptive digital solutions for passenger rail services. However, the study does not focus on AI-assisted generation of wagon interface configurations.

The application of digital twins for railways, as described in paper [16], focuses primarily on tracks, civil structures, rolling stock, overhead lines, predictive maintenance, monitoring, and infrastructure management. This confirms that passenger-facing booking interfaces remain underrepresented in study on railway digitalization.

Paper [17] shows that practical LLM performance depends on workflow design. This is relevant because wagon schema generation cannot rely on model capability alone. It requires controlled task organization and bounded processing.

Paper [18] demonstrates that visual input can be converted into structured output. This supports the use of visual-to-schema processing. At the same time, generating train car schematics is more specialized than screenshot analysis. It requires domain-specific notation for JavaScript objects, with preserved seat numbers, element types, directional attributes, accessibility features, and convenience-related parameters.

Paper [19] confirms that LLM-based agents can support technical workflows. Paper [20] describes the main functions of autonomous agents, including planning, execution, tool usage, and memory management. These functions are relevant to the proposed two-level architecture.

A limitation of these studies is their general scope. They do not provide a workflow for verified wagon interface schemas or deterministic validation in railway booking systems. This study fills this gap by combining limited-context schema processing, visual schema generation, and pre-production validation within the constraints of the railway interface.

The results presented in [21] show that chain-of-thought prompting can improve performance in complex reasoning tasks. This is relevant because the generation of wagon schemas involves spatial reasoning, numbering dependencies, and structural constraints. However, improving reasoning alone is insufficient for implementation in a production environment. On a railway booking platform, the generated artifact must undergo formal validation and human visual inspection before it can affect passengers. Therefore, reasoning must be separated from the direct execution of the configuration and integrated into a controlled architecture.

Another constraint concerns the management of context windows. In [22], it is shown that language models are few-shot learners limited by finite context windows. This limitation is significant for managing wagon configurations, as a national railway booking system may contain over 150 wagon configurations. Loading the entire set of configurations into a single prompt can increase context complexity, computational costs, and the likelihood of inappropriate generation.

Retrieval-Augmented Generation proposed in [23] can expand access to relevant knowledge without directly increasing the prompt size. However, retrieval does not fully solve the problem of modifying large structured artifacts. The resulting fragment may be relevant, but the model must still modify only the correct area of the schema and preserve all unchanged parts of the configuration. Therefore, a more suitable approach for generating wagon schemas is context-limited processing, in which the active context is restricted to the specific area of the schema required for the current operation.

The risk of hallucinations is another key issue. In [24], hallucinations in natural language generation are systematized as a serious reliability problem. [25] discusses methods

for mitigating hallucinations in large language models. These studies demonstrate that the generated output can be plausible but incorrect. This is particularly dangerous for railway booking interfaces, where a plausible but invalid schema can distort the actual wagon layout.

In [26], a validation-oriented method for hallucination detection is proposed. In [27], the use of external tools for deterministic computation is demonstrated. These approaches show that model outputs can be verified and constrained. However, hallucination mitigation in text generation does not automatically ensure production schema safety. Railway booking systems require that invalid AI-generated schemas never reach production. This requires deterministic schema validation, human visual approval, and revision history, rather than reliance on model output alone.

The deployment model is also important. In [28], serverless computing is described as suitable for event-driven workloads. In [29], analysis of real-world serverless workloads confirms that intermittent and bursty tasks can benefit from this model. Paper [30] examines current trends and open problems in serverless computing, including AI-related scenarios. These studies are relevant because wagon schema updates are not continuous high-throughput operations. They occur irregularly, but must be processed quickly and safely.

However, most research on serverless artificial intelligence focuses on general cloud workloads. It rarely addresses low-frequency and high-risk railway configuration tasks. One likely reason is that practical evaluation requires access to real production infrastructure. This makes laboratory replication difficult. One way to address this limitation is to study a system deployed in a real national railway booking environment. In such an environment, update time, engineering cost, deployment complexity, and production safety can be evaluated directly.

Thus, the reviewed literature shows that railway digitalization, large language models, multimodal extraction, digital twins, agentic workflows, context management, hallucination mitigation, and serverless deployment provide separate technological foundations for the studied problem. However, these studies mainly focus on traffic management, maintenance, monitoring, infrastructure, general code generation, or open-ended agentic workflows rather than passenger-facing railway booking interfaces.

As a result, the representation of wagon configurations for railway booking systems remains insufficiently studied. This unresolved problem is practically significant because booking platforms must accurately reflect seat numbering, spatial layout logic, amenities, accessibility elements, and direction-of-travel indicators. In production railway systems, schema errors may affect passenger-facing transactions and cannot be treated as ordinary interface inconsistencies.

Ukrzaliznytsia operates a national booking platform supporting more than 150 wagon configurations across different train classes. This environment demonstrates the practical scale of the unresolved problem because wagon schemas must be updated, validated, versioned, and delivered to production without disrupting the booking interface.

Therefore, the common unresolved problem is how to convert natural-language requests and manufacturer schematics into valid, versioned, and release-independent wagon interface schemas while maintaining bounded context complexity and preventing invalid artificial intelligence-generated outputs from reaching production. Solving this problem requires an artificial intelligence-assisted workflow that combines multimodal schema generation, bounded-context processing, deterministic validation, human visual review, and release-independent content delivery.

3. The aim and objectives of the study

The aim of the study is to develop an artificial intelligence-assisted workflow for automated generation and validation of wagon user interface schemas in a national railway booking system, with a focus on improving operational efficiency and production safety through a two-layer agentic mechanism using multimodal large language models.

The practical use of the obtained results will allow railway booking platform administrators to create, modify, validate, and deploy wagon configurations for web and mobile interfaces without direct editing of JavaScript Object Notation or TypeScript code and without waiting for scheduled software releases. This will reduce the duration and cost of wagon configuration updates, decrease dependence on engineering teams, support artificial intelligence-assisted processing of manufacturer schematics, and prevent invalid artificial intelligence-generated schemas from reaching passenger-facing booking interfaces.

To achieve the aim, the following objectives were set:

– to formalize the production problem of manual wagon configuration development for railway booking websites and mobile applications and derive system requirements for artificial intelligence-assisted schema generation;

– to substantiate and implement the functional separation between the Reasoning Engine and the Worker Agent as a mechanism of metadata-level request analysis, scoped schema execution, and bounded-context processing;

– to develop a multimodal visual-to-schema pipeline for transforming manufacturer schematics into template-constrained JavaScript Object Notation configurations while preserving seat numbering, spatial layout, amenities, accessibility elements, and direction-of-travel indicators;

– to compare the developed artificial intelligence-assisted workflow with the previous manual workflow according to update time, engineering cost, deployment coupling, required technical expertise, scalability, and production safety.

4. Materials and methods

The object of this study is the process of automated generation and validation of structured wagon user interface schemas for national railway booking platforms. The principal hypothesis of this study is that decomposing complex schema generation into a separate reasoning layer operating on metadata and an execution layer operating on content within a bounded context can preserve bounded context complexity regardless of the volume of wagon configurations and prevent invalid generated schemas from reaching production through deterministic validation and human review.

During the study, a number of presumptions were agreed upon:

1) it will be easier to target particular schema regions if administrator appeals will be placed into a discrete schema modification;

2) a hierarchical JavaScript Object Notation schema contract is suitable for representing wagon configurations;

3) manufacturer-provided schematics include sufficient visual data to build seat layouts and placement of amenities.

The following simplifications were adopted:

1) the evaluation is conducted on a single railway operator's schema format (Ukrzaliznytsia);

2) only one multimodal model, Google Gemini 1.5, developed by Google LLC, United States, was used as a system component for schema extraction and generation;

3) the quantitative evaluation is based on operational metrics rather than controlled laboratory experiments.

The methods of the study included system analysis and formalization for designing the architecture, comparative operational evaluation against the manual baseline, and visual validation by human administrators before production release.

The evaluation criteria were the time required to create a new wagon type, monthly engineering cost, deployment coupling, required technical expertise, scalability of configuration management, schematic processing outcomes, and the number of invalid AI-generated outputs reaching production. These indicators were compared with the pre-intervention manual baseline.

The system analyzed in this study was architected and deployed during the author's consulting engagement on the Ukrzaliznytsia booking platform in 2022–2024. The empirical basis of the study included the deployed production environment, managed wagon configurations, and manufacturer schematics processed through the system during the evaluation period. The software environment used in the study included Google Gemini and Google Cloud Platform services, both developed by Google LLC, United States. The frontend was implemented using TypeScript and VueJS, open-source software technologies originally developed by Microsoft, United States, and Evan You, China, respectively. The railway booking platform and wagon configuration data belonged to Public JSC "Ukrainian Railways" (Ukrzaliznytsia), Ukraine.

The deployed system operated on the Ukrzaliznytsia booking platform (https://booking.uz.gov.ua/), which served more than 20 million passengers in 2024 [1]. According to platform statistics, approximately 86% of railway tickets were purchased online in 2024 [2], making the booking platform the primary passenger-facing digital channel of the national railway. The system managed more than 150 wagon configurations across multiple train classes. This deployment context was used as the empirical basis for comparing the artificial intelligence-assisted workflow with the previous manual workflow.

The evaluation was conducted in several stages. At the first stage, the manual workflow for wagon configuration development was analyzed in order to identify the production bottleneck and derive system requirements. At the second stage, the two-layer agentic architecture and the multimodal visual-to-schema pipeline were implemented in the deployed environment. At the third stage, the developed workflow was compared with the previous manual workflow according to update time, monthly engineering cost, deployment coupling, required technical expertise, scalability of configuration management, schematic processing outcomes, and production safety.

The system was implemented on Google Cloud Platform using a serverless architecture [28]. Google Gemini 1.5, developed by Google LLC, United States, was used as the multimodal component for text and image processing [9]. Google Cloud Storage was used for versioned schema storage and revision history. The frontend was implemented using TypeScript and Vue.js. A custom JavaScript Object Notation Schema validator was used to check structural correctness and domain-specific wagon configuration rules.

The manual workflow served as the baseline condition, while the artificial intelligence-assisted workflow served as the intervention condition for evaluating operational effectiveness and production safety.

5. Results of implementing the artificial intelligence-assisted wagon schema generation workflow

5. 1. Results of formalizing the production problem and system requirements

The production process was first formalized as a sequence of administrator input, reasoning, schema execution, validation, storage, and production delivery. Fig. 1 provides a general map of this workflow and shows where invalid generated schemas are blocked before release.

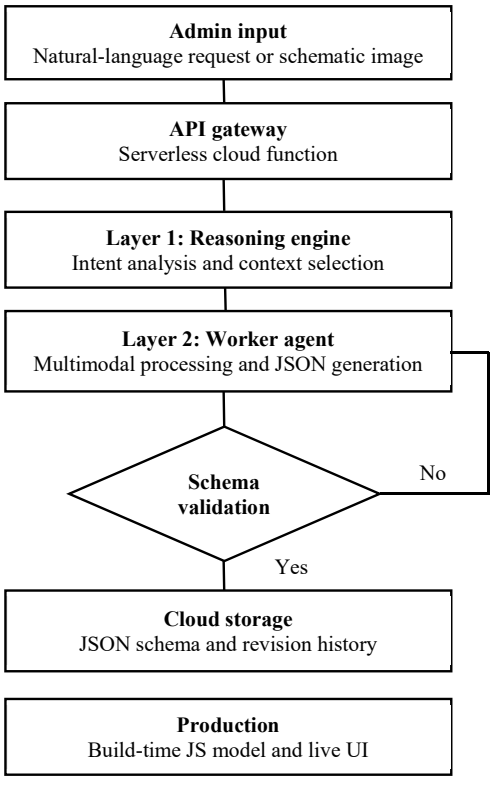


Fig. 1. General workflow of the Wagons Lab system showing the data flow from administrator input to validated production delivery

Let's note that validation is placed before cloud storage and production delivery. This prevents invalid generated schemas from entering the version history or reaching the passenger-facing interface.

In the Ukrzaliznytsia online booking platform, a wagon functions as an interactive interface, not as a fixed visual image. The interface is generated from a structured JavaScript Object Notation schema. This schema defines the essential layout data, including seat positions, numbering, travel direction, accessibility elements, luggage areas, and amenities. Fig. 2, *a* shows an example fragment of this schema. Each element in the schema is linked to a TypeScript object used by the front-end renderer to build the seat selection interface. Fig. 2, *b* presents the corresponding visual output in the booking system.

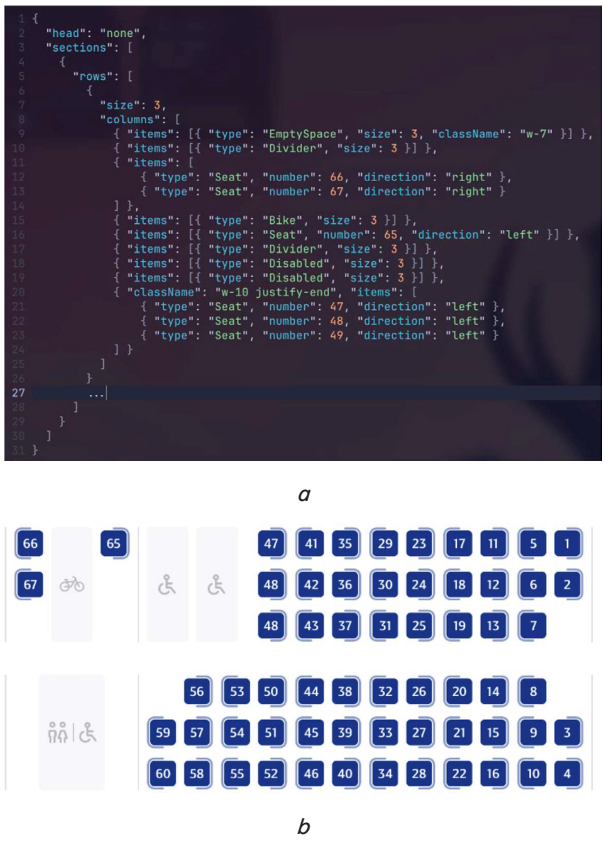


Fig. 2. Representation of the wagon interface structure: *a* — example of a wagon JavaScript Object Notation schema fragment showing seat positions, types, direction attributes, and amenity placements; *b* — visual representation of the schema

Let's note the hierarchy: wagon → sections → rows → columns → items. Each seat also has fixed attributes. Together, these attributes form a schema contract that the LLM fills in but cannot freely change.

The schema follows a nested structure. A wagon consists of sections. Sections include rows, rows include columns, and columns contain individual items. These items may be seats with number, direction, and class attributes. They may also include accessibility elements, such as wheelchair spaces and accessible restrooms, service elements, such as staff compartments and luggage racks, and amenities, such as toilets or bicycle storage areas. Because these elements are connected, the schema must preserve numbering order, spatial placement, and accessibility rules.

Before deployment, wagon schema updates were handled manually and tied to the software release cycle. Site administrators first sent requested changes to the engineering team. The engineers would then manually develop or adapt the JSON structures. Domain knowledge of the wagon plan and technical expertise regarding the schema contract are essential aspects for quality work, as soon as the schema needs to be verified against the rendering output by means of visual comparison. The last step is to deploy and to update the code to production as part of a scheduled software release conducted by a team.

This process was only possible during software deployment and resulted in several considerable inefficiencies. It usually took 24 hours to produce a new wagon, and the manual JSON sequence revealed inconsistencies in the formatting, which required iterative quality control cycles. A dedicated engineering team was required to manage over 150 wagon configurations costing thousands of dollars per month. These constraints provoked the four system requirements mentioned in Table 1. Therefore, the formalized production problem was defined as the need to convert natural-language requests and manufacturer schematics into valid, versioned, and release-independent wagon interface schemas under requirements R1–R4.

Table 1

System requirements derived from the engineering bottleneck analysis

ID	Requirement	Description
R1	Content-code decoupling	Wagon schema definitions must be updatable independently of application releases
R2	Natural language interface	Non-technical administrators must specify schema modifications without JSON syntax knowledge
R3	Visual input support	System must accept manufacturer wagon schematics (PDF/JPEG) and translate them into valid schemas
R4	Production safety	No AI-generated schema may reach production without deterministic validation and human review

This baseline workflow served as the reference condition against which the operational performance of the proposed architecture was subsequently evaluated.

5. 2. Results of substantiating reasoning-execution separation and bounded-context schema processing

Processing complex wagon schemas in a single LLM prompt is both inefficient and error-prone, as large schemas may overflow available context windows [22]. This constraint was addressed by separating the task into metadata-level reasoning and bounded-context schema execution. Fig. 3 shows the functional separation between metadata-level reasoning and bounded-context schema execution.

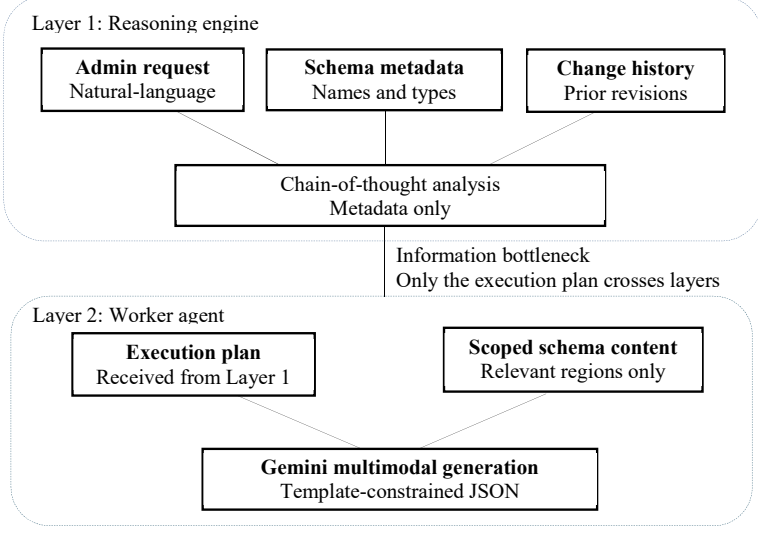


Fig. 3. Functional separation between metadata-level reasoning and bounded-context schema execution: Layer 1 produces a scoped execution plan from a natural-language request, and Layer 2 applies this plan only to the relevant schema fragment

Let's note the explicit information bottleneck between Layer 1 and Layer 2: only the scoped execution plan crosses this boundary, which is the mechanism that maintains bounded context complexity.

Layer 1: Reasoning Engine. This layer processes administrator requests in natural language, such as "Update second-class seats and add wheelchair accessibility to seats 47–48." It performs intent and scope analysis without directly modifying existing components. It identifies the minimum required data: relevant schema parts, element definitions, and change history. Irrelevant content is excluded to reduce redundancy.

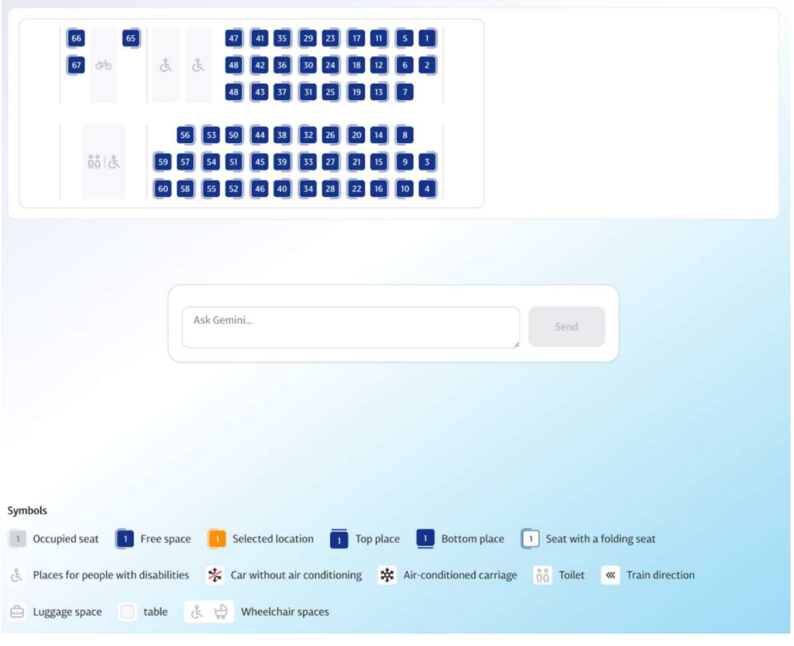
Layer 2: The Worker Agent. This layer receives the structured execution plan and applies it only to the relevant schema fragments. It follows the specified constraints and checks seat numbering, spatial constraints, and amenity placement before generating the final JSON output.

Before schema generation, this layer uses structured intermediate reasoning steps to check seat numbering, spatial constraints, and amenity placement [21]. For example, when accessibility spaces are added, the model must justify adjacent seat numbering, path width constraints, and amenity placement rules.

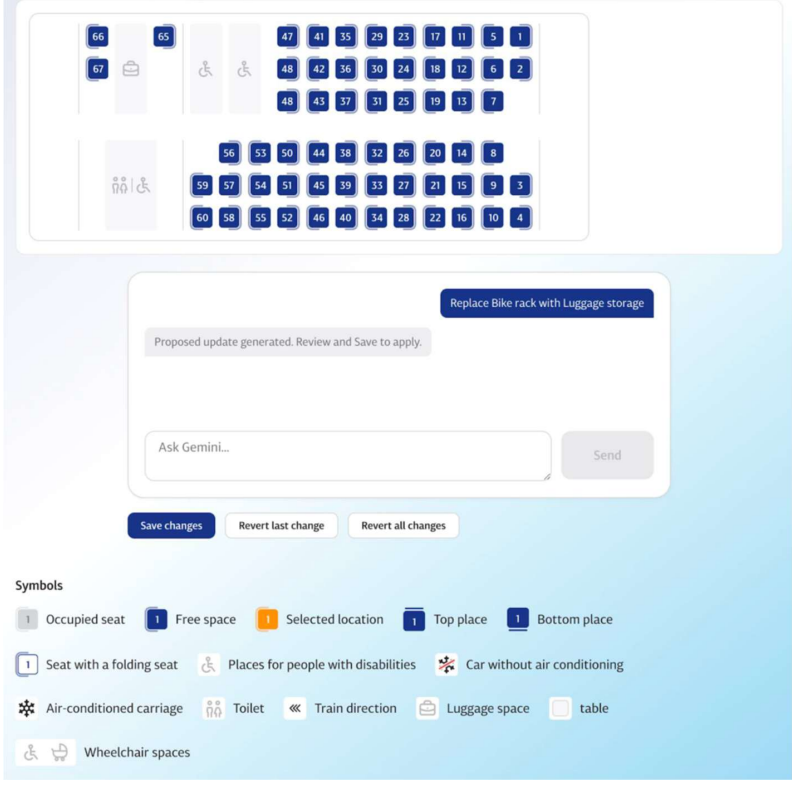
To meet requirement R1, the wagon schema is stored separately from the application codebase as a configuration file in a Google Cloud Storage bucket. During build time, the frontend fetches the latest configuration and generates a TypeScript wagon model to avoid runtime JSON parsing overhead. This architecture turns wagon updates from a release-dependent activity into a content management task. Administrators can update wagon configurations and see the changes on the production booking platform without software releases, code reviews, or deployment pipelines.

The Ukrzaliznytsia booking platform successfully integrated and deployed the two-layer agentic architecture. The Reasoning Engine, Layer 1, accurately parses natural language requests into structured execution plans for all tested request types, including reordering, availability updates, wagon cloning, and new wagon generation. The Worker Agent, Layer 2, executes these plans within a bounded context window without loading all 150+ configurations simultaneously.

In Edit Mode, requirement R2, administrators interact with wagon configurations using natural language commands. For example, the instruction "Copy wagon configuration #23, but replace the bike rack with luggage storage" starts the following process. The Reasoning Engine analyzes the request and identifies cloning of the existing configuration, locating the terminal row, and replacing the bike rack with luggage storage. The initial configuration of wagon #23 is shown in Fig. 4, *a*. The Worker Agent then performs these operations on the actual schema and uses structured constraint checking to account for structural effects, column width adjustments, and amenity placement [21]. It then generates the final JSON. Fig. 4, *b* shows the modified configuration with luggage storage in the last row.



a



b

Fig. 4. Before-and-after comparison: *a* — original wagon configuration #23; *b* — modified configuration with luggage storage in the last row

The continuity of seat numbering in the unmodified rows and the structural integrity of the modified terminal row show that Layer 2 correctly applies the amenity substitution and spatial constraints specified by Layer 1.

The main architectural result is that the context window size remains constant regardless of the total number of wagon configurations in the system. Whether the system manages 50 or more configurations, the Reasoning Engine processes only metadata, including schema section names, element types, and change history. The Worker Agent receives only the specific schema regions defined in the execution plan.

Thus, the functional separation between the Reasoning Engine and the Worker Agent limits processing to specific schema regions, reduces context complexity across more than 150 configurations, and preserves the structural integrity of modified wagon layouts. The metadata-to-plan transition in Fig. 3 and the scoped modification example in Fig. 4 demonstrate this result.

5. 3. Results of implementing the multimodal visual-to-schema pipeline

The multimodal visual-to-schema pipeline was implemented to transform manufacturer schematics into structured JavaScript Object Notation configurations. Railway operators typically provide wagon specifications in PDF format or as schematic JPEG format. The "visual-to-schema" pipeline processes requirement R3, converting visual artifacts into structured configurations. The pipeline workflow is represented in Fig. 5.

Let's note that step (c) constrains the large language model to populate predefined template fields rather than generating JSON freely, this is the mechanism that prevents structural hallucinations.

Create Mode is utilized by the administrator to upload the wagon schematic image. With the help of Google Gemini vision capabilities [9], the system extracts seat layouts and numbering sequences, spatial arrangement (head, rows, columns, positions), direction-of-travel indicators, and amenities (restrooms, conductor compartments, bicycle storage). The visual items are mapped to the unified JSON schema structure via a template-driven creation process. The model does not generate the full schema freely; it fills predefined template fields with values extracted from the visual input. The template constraints ensure that the result will always match the anticipated contract schema, regardless of the complexity or uniqueness of the schematic input.

The Create Mode user interface is shown in Fig. 6, *a*. The image is sent to Google Gemini together with a structured prompt and an explicit specification of the output schema contract. The model extracts spatial layout and amenity locations from the visual input, and the result is then processed to verify schema conformance. An example of the manufacturer wagon schematic used as input is shown in Fig. 6, *b*. The administrator reviews the rendered component and may refine it with additional natural-language prompts. The generated wagon component rendered from the extracted schema is shown in Fig. 6, *c*.

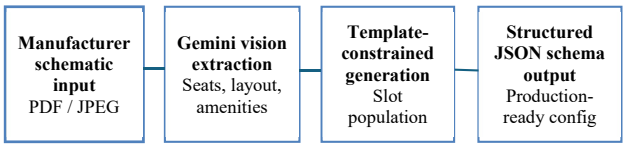


Fig. 5. Visual-to-schema pipeline workflow showing four consecutive stages: manufacturer schematic input, Gemini vision extraction, template-constrained generation, and structured JavaScript Object Notation schema output

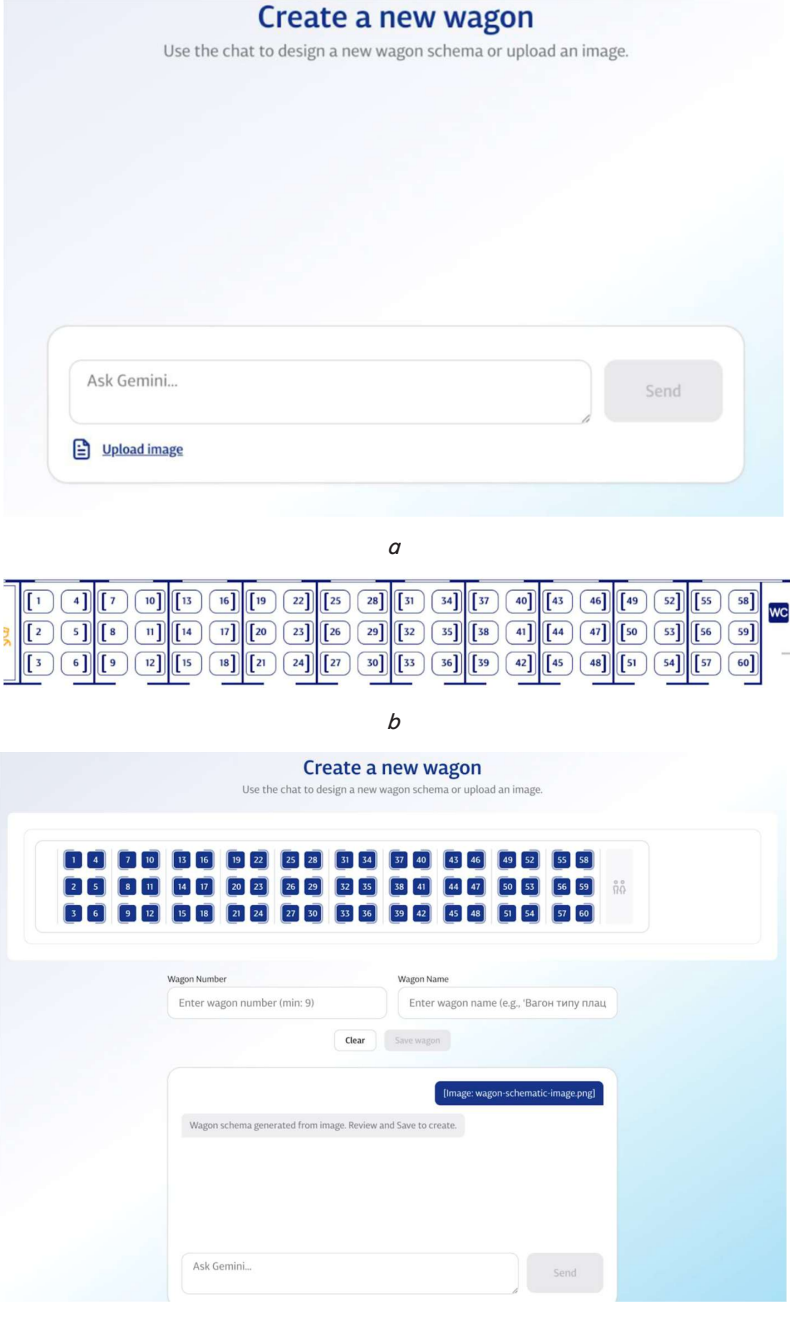


Fig. 6. Visual schema creation example: *a* — user interface; *b* — manufacturer wagon schematic (input); *c* — artificial intelligence-generated wagon component rendered from the extracted schema (output)

Let's note the correct preservation of seat numbering sequence, the accurate placement of amenities (restrooms and luggage storage), and the consistent direction-of-travel indicators between the input schematic and the rendered output.

The multimodal pipeline successfully processed schematics for all tested wagon types across multiple train classes, including first-class sleeper wagons, second-class compartment wagons, third-class open wagons, luxury wagons, and special-purpose wagons with non-standard accessibility configurations. Across the 20 schematics processed during the evaluation period, only one required manual correction after a full generation pass, while the others were refined through one or two follow-up natural-language prompts. The template-constrained generation method ensured that all outputs adhered to the JSON schema contract, regardless of the input quality. In cases where the initial extraction was incomplete or inaccurate, administrators refined the output using follow-up natural-language prompts within the same two-layer workflow, where one layer interpreted the requested correction and the other applied it to the relevant schema fragment.

5. 4. Results of comparing operational effectiveness and production safety with the manual baseline

The developed architecture was compared with the manual workflow previously used on the Ukrzaliznytsia booking platform. The evaluation covered two groups of indicators: operational effectiveness and production safety. Operational effectiveness was assessed by update time, monthly engineering cost, deployment coupling, required technical expertise, scalability of configuration management, and schematic processing results. Production safety was assessed by the number of invalid artificial intelligence-generated schemas reaching production and the number of production incidents caused by model hallucinations. The evaluation results are presented in Table 2.

As shown in Table 2, the developed architecture improved both operational effectiveness and production safety compared with the manual baseline. The time needed to create a new wagon type decreased from approximately 24 hours to approximately 15 minutes, which corresponds to a 99% reduction. Monthly engineering involvement was replaced by serverless infrastructure costing less than 5 USD per month. Schema updates also became independent of software release cycles. Across 20 processed manufacturer schematics, 19 cases required one or two follow-up natural-language prompts, while one case required manual correction after the full generation pass. No invalid artificial intelligence-generated schemas reached production, and no production incidents caused by model hallucinations were recorded.

Schema verification. The automated JSON Schema validator intercepted and blocked all structurally incorrect schemas created by the Worker Agent, preventing any hallucinations generated by artificial intelligence from appearing in the revision history. The structurally incorrect schemas included: incorrect nesting depth, missing required fields, duplicate seat numbers, and elements that were spatially incompatible. All of these issues were rejected before reaching the visual review phase.

Visual verification. Even with minimal training, site administrators successfully verified that the generated schemas met the input data requirements. The natural language interface reduced the learning period from several weeks (editing JSON) to just a few hours (for conversational interactions). The administrators noted that the visual validation step, reviewing the visualized component before approval, gave them confidence in the results generated by artificial intelligence and reduced their concerns about deploying AI-generated configurations in the production environment.

Revision history. The version-based data storage mechanism enabled administrators to quickly restore a previous state when they identified semantic (rather than structural) issues in approved configurations. Though there was no need to apply rollbacks because of hallucinations, the tool was tested with changes to the administrative settings, confirming that it was ready for operation.

Table 2

Evaluation of operational effectiveness and production safety of the developed architecture

Evaluation criterion	Manual baseline	Developed architecture	Result
Time to create a new wagon type	Approximately 24 hours	Approximately 15 minutes	99% reduction
Monthly engineering cost	Thousands of USD per month	Less than 5 USD per month for serverless infrastructure	Approximately 99% reduction
Deployment coupling	Release-bound software update	Release-independent content update	Fully decoupled from software releases
Required technical expertise	JavaScript Object Notation and TypeScript editing	Natural-language interaction and visual review	Access shifted to non-technical administrators
Number of managed wagon configurations	More than 150 configurations managed manually	More than 150 configurations managed through scoped schema processing	Full configuration set supported without loading all schemas into one context
Number of manufacturer schematics processed	Not applicable to the manual baseline	20 schematics from different wagon classes	19 cases required one or two follow-up prompts; 1 case required manual correction
Invalid artificial intelligence-generated schemas reaching production	Not applicable to the manual baseline	0 cases	Invalid outputs blocked before production
Production incidents caused by model hallucinations	Not applicable to the manual baseline	0 incidents	Zero production incidents caused by hallucinations
Administrator training effort	Several weeks for manual schema editing	Several hours for conversational interaction and visual review	Training effort reduced from weeks to hours

Scalability of the validation pipeline. The serverless architecture automatically scaled to meet demand. No performance degradation was noted. The number of supported wagon types increased since adoption of this pipeline, and the zero-deploy content delivery mechanism enabled the implementation of validated configurations on the production platform within minutes of approval.

6. Discussion of the two-layer architecture performance and applicability

The obtained results show that wagon schema management can be transformed from an engineering-dependent process into a content-driven workflow. This transformation is reflected in Table 2, where the time required to create a new wagon type decreased from approximately 24 hours to 15 minutes, monthly infrastructure cost decreased to less than 5 dollars, deployment coupling changed from release-bound to independent, and no invalid artificial intelligence-generated outputs reached production. These results can be explained by the architecture shown in Fig. 1: administrator input is processed through the API gateway, divided between the reasoning and worker layers, checked by schema validation, and only then stored and delivered to production. Thus, the performance improvement results not only from model generation, but from the combined effects of output separation, large-scale schema processing, natural language interaction, deterministic validation, and content propagation without code deployment.

These results confirm the study's main hypothesis. The separation of schema generation into reasoning and execution layers kept contextual complexity bounded for more than 150 wagon configurations. Deterministic validation and human visual inspection prevented invalid model outputs from entering production. In practice, non-technical administrators can manage wagon configurations outside software release cycles. This reduces update time, operational costs, and the risk of invalid production results.

The results of the study reveal several unique features. First, the developed process integrates metadata-level reasoning and bounded-context template execution into a single production workflow. Second, multimodal extraction is based on constrained schema generation using predefined JavaScript Object Notation structures, rather than free-form generation. Third, code-independent content distribution is combined with deterministic validation, human visual review, and edit history. These features distinguish this approach from code generation methods based on large language models and from railway digital twin methods, since the generated artifacts are not only produced but also verified and version-controlled before production, and blocked if their structure is invalid.

The results also demonstrate that the proposed architecture extends study in railway digitalization and digital twins [12–14]. Current study on railway digital twins is primarily focused on predictive maintenance, infrastructure monitoring, and operational control. In contrast, this study applies the logic of digital representation to a passenger-oriented booking interface. Wagons are considered not only as physical objects but also as structured digital artifacts that must support seat numbering, spatial logic, amenities, accessibility features, and visual consistency. Fig. 2 illustrates this point. It shows a fragment of the JavaScript Object Notation (JSON) schema for the same wagon and its corresponding user interface. The figure explains why a digitized model must preserve not only visual similarity but also its internal hierarchical structure, including sections, rows, columns, elements, and seat attributes.

The results also highlight the importance of separating intent from execution. As shown in Fig. 3, the first layer handles administrator requests, schema metadata, and change history, while the second layer receives only execution plans and the schema's scoped content. This information bottleneck explains why the system can manage more than 150 wagon configurations without loading all layouts into a single model context. Fig. 4 demonstrates the same mechanism: natural language commands modify only the relevant terminal rows, while preserving seat-numbering continuity and spatial areas. Therefore, as the number of configurations increases, context complexity remains low.

The proposed architecture also differs from visual structure extraction methods such as Pix2Struct [18]. In these methods, the primary task is to transform the visual input into a structured representation. In this study, visual extraction is only part of a broader manufacturing workflow, as shown in Fig. 5. First, the Gemini visual analysis component processes the manufacturer's schematic diagram, then transforms it by generating template constraints, and finally produces a structured output template. Fig. 6 further illustrates why visual extraction alone is insufficient: the generated interface must be compared to the manufacturer's schematic diagram, preserving seat numbering, amenity placement, and directional indicators. Therefore, the resulting output must comply with the formal template contract, be verified via deterministic validation, support human validation, and be delivered to the booking platform without a software release.

Compared to agent approaches based on large language models [19, 20], the proposed architecture meets stricter production requirements. General proxy systems are typically evaluated in open environments, and their success depends on task completion. In this study, simple task completion is insufficient. The generated template must be formally valid, visually accurate, and safely deployable in the national railway reservation system. This explains why strict inter-level information bottlenecks are more appropriate than extended multi-agent interaction schemes. As shown in Fig. 3, this bottleneck also reduces the likelihood of uncontrolled schema changes, since the working level receives only the relevant schema regions selected by the reasoning layer, rather than the full configuration library.

Compared to retrieval-augmented generation [23], the proposed method does not rely on linear growth in context size. Methods with improved search can provide relevant external information, but they cannot automatically solve the problem of modifying large structured artifacts. In the proposed architecture, only the relevant areas of the templates are processed. Fig. 3 illustrates this principle at the architectural level, and Fig. 4 at the interface level, where only selected segments of the train cars are modified, while the rest of the configuration remains unchanged. This reduces the contextual load and the risk of inconsistencies.

Compared to single-prompt large language modeling methods [24], which are prone to hallucinations in complex generation tasks, the proposed system employs deterministic validation prior to deployment. No production incidents caused by hallucinations were detected. This result does not

mean that the model never produced erroneous outputs, but rather that invalid outputs were effectively prevented from entering the production environment thanks to the validation path shown in Fig. 1, the template constraint generation stage shown in Fig. 5, and the visual review by an administrator shown in Fig. 6.

The results regarding costs are consistent with studies on serverless computing [28–30], which show that periodic and event-driven workloads can benefit from serverless deployments. However, this case differs from typical serverless AI scenarios. Generating wagon schemas is not a high-throughput consumer-level task. This explains why the cost impact in Table 2 is primarily related to eliminating the constant involvement of engineers and version-controlled deployments, rather than optimizing large-scale computational performance. Instead, wagon schema generation is a low-frequency but high-risk operation, and every update must be accurate, traceable, and safe to ensure usability in a production environment.

Furthermore, the proposed solution has clearly defined use cases. The results are most reproducible in environments where a wagon layout can be represented as a hierarchical schema, as shown in Fig. 2, and where operational requirements are similar to those listed in Table 1. These requirements include content-code decoupling, a natural language interface, support for visual input, and deterministic production safety. Migration to other railway operators is possible, but it will require adaptation of schema contracts, validation rules, interface logic, and local booking constraints.

A limitation of this study is the empirical scope of the evaluation. The results are based on only one national railway booking platform, one operator-specific schema format, one core server-side component of the multimodal model, and 20 manufacturer schematics processed during the evaluation period. These limitations do not affect the practical applicability of the results presented in Table 2, but they limit the extent to which the same quantitative results can be generalized to other railway operators, schema standards, or model server components. Therefore, future study should include controlled comparative testing of multiple multimodal models, multiple railway schema formats, and a larger number of manufacturer schematics.

The proposed architecture has shortcomings that differ from these study limitations. First, the system depends on the quality and completeness of the manufacturer's schematics. If the input images are incomplete, outdated, or visually ambiguous, the generated schemas may require additional correction by an administrator. These shortcomings can be mitigated by adding automatic image quality checks and visual detection of discrepancies between the input schematics and the rendered output. Second, the architecture relies on predefined schema contracts. This improves safety but limits flexibility when new wagon components appear. This can be addressed by expanding the schema library and validation rules before adding new component categories. Third, the architecture still requires human visual inspection because formal validation cannot detect all semantic differences between physical wagons and the rendered interface. This can be reduced by combining deterministic validation with automated visual comparison, test rendering, and domain-specific rule checks.

Further study can follow two application paths. The first is real-time wagon reconfiguration during schedule disruptions. This requires integration with real-time operational data, access control rules, and pre-deployment validation. The second is synchronization between the physical state of a wagon and its digital model using sensors or operational data. These directions should address data heterogeneity, update delays, incomplete manufacturer documentation, and operational safety during automated schema changes.

7. Conclusions

1. For national railway booking platforms, the operational challenge of creating schematics for the wagon user interface was formalized. The analysis showed that manually updating a schema takes about 24 hours for each configuration, depends on software release cycles, requires knowledge of JavaScript Object Notation and TypeScript, and is prone to production errors. Four system requirements were identified: separation of content from code, natural language interaction, support for visual input, and production safety through deterministic validation and human review.

2. A functional separation between the reasoning engine and the Worker Agent was established and implemented as a schema processing mechanism in a restricted context. The reasoning engine performed request analysis at the metadata level and generated high-level execution plans, while the Worker Agent modified only the relevant schema fragments. This mechanism supported work with more than 150 wagon configurations without loading the entire set of schematics into a single model context.

3. A multimodal visual-diagrammatic pipeline was implemented to convert manufacturer schematics into structured configurations in JavaScript Object Notation. The pipeline processed 20 schemas from different wagon classes. After a full generation cycle, only one diagram required manual correction, while the remaining cases were refined using one or two subsequent iterations.

4. The operational effectiveness and production safety of the artificial intelligence-assisted workflow were evaluated against the manual baseline. The time needed to create a new wagon type decreased from approximately 24 hours to approximately 15 minutes, corresponding to a 99% reduction. Monthly engineering cost decreased from thousands of dollars to less than 5 dollars for serverless infrastructure. No invalid artificial intelligence-generated schemas reached production, and no production incidents caused by model hallucinations were recorded.

Conflict of interest

The author declares that there is no conflict of interest in relation to this study, whether financial, personal, authorship or otherwise, that could affect the study and its results presented in this paper.

Financing

The study was performed without financial support.

Data availability

Data cannot be made available for reasons disclosed in the data availability statement. The wagon schema configurations are proprietary to Public JSC "Ukrainian Rail-

ways" (Ukrzaliznytsia). The developed architecture, methodology, and evaluation logic are disclosed in sufficient detail to support replication in analogous domains.

Use of artificial intelligence

Google Gemini 1.5, developed by Google LLC, United States, was used only as a functional component of the AI Wagons Lab system described in Sections 4, 5.2, and 5.3. The model was used for multimodal extraction of seat layouts, numbering, direction indicators, and amenities from manufacturer wagon schematics, and for template-constrained generation of wagon schemas in JavaScript Object Notation format.

The model was not used to write, paraphrase, translate, shorten, expand, or stylistically edit any part of the manuscript text, including the abstract, introduction, literature review, materials and methods, results, discussion, conclusions, references, or author statements.

The outputs generated by the AI Wagons Lab system were checked through deterministic schema validation, visual review by administrators, revision history, and comparison with the manual baseline workflow. Invalid schema outputs were blocked before production release. The model-generated outputs did not independently determine the study conclusions. The conclusions were formulated by the author on the basis of validated operational results obtained in the deployed railway booking environment.

Acknowledgments

The author thanks the engineering team at Public JSC "Ukrainian Railways" (Ukrzaliznytsia) for their collaboration during system deployment and evaluation.

Author's contributions

Ivan Dobrovolskyi: Conceptualization, Methodology, Software, Validation, Investigation, Writing – Original Draft, Writing – Review & Editing, Visualization, Project administration.

References

1. Dpty PM Kuleba announces record transportation of Ukrzaliznytsia in 2024. Interfax-Ukraine. Available at: https://en.interfax.com.ua/news/economic/1034288.html
2. According to the results of 2024, 86% of railway tickets were purchased online (2025). Ukrzaliznytsia. Available at: https://t.me/UkrzalInfo/6461
3. Berrios Villalba, A. (2020). How to Speed Up Digitization in the Railway [Viewpoint]. IEEE Electrification Magazine, 8 (1), 76–75. https://doi.org/10.1109/mele.2019.2962895
4. Cecchetti, G., Lina, A., Ruscelli, Ulianov, C., Hyde, P., Liu, J. et al. (2023). Toward new generation railway Traffic Management Systems: the contribution of the OPTIMA project. Transportation Research Procedia, 72, 3166–3173. https://doi.org/10.1016/j.trpro.2023.11.882
5. Bezuidenhout, M., Jooste, J. L., Lucke, D., Fourie, C. J. (2023). Leveraging digitilisation and machine learning for improved railway operations and maintenance. Procedia CIRP, 120, 702–707. https://doi.org/10.1016/j.procir.2023.09.062
6. Jiang, J., Wang, F., Shen, J., Kim, S., Kim, S. (2026). A Survey on Large Language Models for Code Generation. ACM Transactions on Software Engineering and Methodology, 35 (2), 1–72. https://doi.org/10.1145/3747588
7. Chen, M., Tworek, J., Jun, H., Yuan, Q., Pinto, H. P. d. O., Kaplan, J. et al. (2021). Evaluating large language models trained on code. arXiv. https://doi.org/10.48550/arXiv.2107.03374
8. Anil, R., Borgeaud, S., Alayrac, J.-B., Yu, J., Soricut, R., Schalkwyk, J. et al. (2023). Gemini: a family of highly capable multimodal models. arXiv. https://doi.org/10.48550/arXiv.2312.11805
9. Georgiev, P., Lei, V. I., Burnell, R., Bai, L., Gulati, A., Tanzer, G. et al. (2024). Gemini 1.5: unlocking multimodal understanding across millions of tokens of context. arXiv. https://doi.org/10.48550/arXiv.2403.05530
10. Pasichnyk, V., Horlatch, V. (2025). Automated extraction of key parameters and detection of inconsistencies in clinical documentation using large language models. Eastern-European Journal of Enterprise Technologies, 6 (2 (138)), 6–14. https://doi.org/10.15587/1729-4061.2025.337915
11. Zhuravchak, A., Piskozub, A., Skorynovych, B., Lakh, Y., Zhuravchak, D., Hlushchenko, P. et al. (2025). Design and development of a large language model-based tool for vulnerability detection. Eastern-European Journal of Enterprise Technologies, 2 (2 (134)), 75–83. https://doi.org/10.15587/1729-4061.2025.325251
12. Sarp, S., Kuzlu, M., Jovanovic, V., Polat, Z., Guler, O. (2024). Digitalization of railway transportation through AI-powered services: digital twin trains. European Transport Research Review, 16 (1). https://doi.org/10.1186/s12544-024-00679-5
13. De Donato, L., Dirnfeld, R., Somma, A., De Benedictis, A., Flammini, F., Marrone, S. et al. (2023). Towards AI-assisted digital twins for smart railways: preliminary guideline and reference architecture. Journal of Reliable Intelligent Environments, 9 (3), 303–317. https://doi.org/10.1007/s40860-023-00208-6
14. Dirnfeld, R., De Donato, L., Somma, A., Azari, M. S., Marrone, S., Flammini, F., Vittorini, V. (2024). Integrating AI and DTs: challenges and opportunities in railway maintenance application and beyond. Simulation, 100 (9), 903–917. https://doi.org/10.1177/00375497241229756
15. Bobyl, V., Matusevych, O., Dron, M., Taranenko, A. (2024). The concept of forming a system of change management in the domain of railroad passenger transportation in Ukraine under the conditions of war. Eastern-European Journal of Enterprise Technologies, 1 (13 (127)), 14–21. https://doi.org/10.15587/1729-4061.2024.297067

16. Thompson, E. A., Lu, P., Alimo, P. K., Atuobi, H. B., Akoto, E. T., Abbew, C. K. (2025). Revolutionizing railway systems: A systematic review of digital twin technologies. High-Speed Railway, 3 (3), 238–250. https://doi.org/10.1016/j.hspr.2025.05.005
17. Zhao, W. X., Zhou, K., Li, J., Tang, T., Dong, Z., Hou, Y. et al. (2026). A Survey of Large Language Models. Frontiers of Computer Science, 20 (12). https://doi.org/10.1007/s11704-026-60308-3
18. Lee, K., Joshi, M., Turc, I., Hu, H., Liu, F., Eisenschlos, J. et al. (2023). Pix2Struct: screenshot parsing as pretraining for visual language understanding. Proceedings of the 40th International Conference on Machine Learning (ICML), PMLR 202. Available at: https://proceedings.mlr.press/v202/lee23g.html
19. Dong, Y., Jiang, X., Qian, J., Wang, T., Zhang, K., Jin, Z., Li, G. (2025). A survey on code generation with LLM-based agents. arXiv. https://doi.org/10.48550/arXiv.2508.00083
20. Wang, L., Ma, C., Feng, X., Zhang, Z., Yang, H., Zhang, J. et al. (2024). A survey on large language model based autonomous agents. Frontiers of Computer Science, 18 (6). https://doi.org/10.1007/s11704-024-40231-1
21. Bosma, M., Chi, E., Ichter, B., Le, Q. V., Schuurmans, D., Wang, X. et al. (2022). Chain-Of-Thought Prompting Elicits Reasoning in Large Language Models. Advances in Neural Information Processing Systems 35, 24824–24837. https://doi.org/10.52202/068431-1800
22. Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P. et al. (2020). Language models are few-shot learners. Advances in Neural Information Processing Systems, 33, 1877–1901. https://doi.org/10.48550/arXiv.2005.14165
23. Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N. et al. (2020). Retrieval-augmented generation for knowledge-intensive NLP tasks. Advances in Neural Information Processing Systems, 33, 9459–9474. https://doi.org/10.48550/arXiv.2005.11401
24. Ji, Z., Lee, N., Frieske, R., Yu, T., Su, D., Xu, Y. et al. (2023). Survey of Hallucination in Natural Language Generation. ACM Computing Surveys, 55 (12), 1–38. https://doi.org/10.1145/3571730
25. Tonmoy, S. M. T. I., Zaman, S. M. M., Jain, V., Rani, A., Rawte, V., Chadha, A., Das, A. (2024). A comprehensive survey of hallucination mitigation techniques in large language models. arXiv. https://doi.org/10.48550/arXiv.2401.01313
26. Manakul, P., Liusie, A., Gales, M. (2023). SelfCheckGPT: Zero-Resource Black-Box Hallucination Detection for Generative Large Language Models. Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, 9004–9017. https://doi.org/10.18653/v1/2023.emnlp-main.557
27. Cancedda, N., Dessi, R., Dwivedi-Yu, J., Hambro, E., Lomeli, M., Raileanu, R. et al. (2023). Toolformer: Language Models Can Teach Themselves to Use Tools. Advances in Neural Information Processing Systems 36, 68539–68551. https://doi.org/10.52202/075280-2997
28. Jonas, E., Schleier-Smith, J., Sreekanti, V., Tsai, C.-C., Khandelwal, A., Pu, Q. et al. (2019). Cloud programming simplified: a Berkeley view on serverless computing. Technical Report No. UCB/EECS-2019-3. Berkeley: EECS Department, University of California. Available at: https://www2.eecs.berkeley.edu/Pubs/TechRpts/2019/EECS-2019-3.html
29. Shahrad, M., Fonseca, R., Goiri, Í., Chaudhry, G., Batum, P., Cooke, J. et al. (2020). Serverless in the wild: characterizing and optimizing the serverless workload at a large cloud provider. 2020 USENIX Annual Technical Conference (USENIX ATC 20), 205–218. Available at: https://www.usenix.org/conference/atc20/presentation/shahrad
30. Baldini, I., Castro, P., Chang, K., Cheng, P., Fink, S., Ishakian, V. et al. (2017). Serverless Computing: Current Trends and Open Problems. Research Advances in Cloud Computing, 1–20. https://doi.org/10.1007/978-981-10-5026-8_1