

A LOW-CODE APPLICATION DEVELOPMENT PLATFORM INTEGRATING AI-ASSISTED DEVELOPMENT

FanHao Ye, ShiWei Xu*

School of Artificial Intelligence, Wenzhou Polytechnic University, Wenzhou 325035, Zhejiang, China.

**Corresponding Author: ShiWei Xu*

Abstract: In the era of large language models, AI-assisted programming has substantially reduced the coding cost of application software, yet difficulties in communicating with clients and inefficiencies in deployment and maintenance persist. Rework and even project failure in real-world projects often stem from discrepancies between the client and the vendor in their understanding of the business, rather than from technology itself. To address this, this paper designs and implements a low-code application development platform that integrates AI-assisted development. Targeting three technical difficulties of current low-code platforms (front-end and back-end fields and rules are hard to configure dynamically, the security of online scripts is hard to constrain, and the reliability of AI-generated output is hard to guarantee), we propose three solutions: (1) a unified metadata model that automatically links front-end and back-end configurations; (2) a sandbox mechanism that restricts data sources, execution permissions and runtime; and (3) purpose-built Skills that constrain the steps and output format of AI generation. The platform is built on an improved RuoYi-Vue-Plus framework with an enhanced permission system, integrates the magic-api script engine, a visual design engine and sandbox technology, and builds its AI-assisted development capability on ByteDance's TRAE. Application in a manufacturing client's MES project shows that the development time of a conventional module fell from one week to 1–2 days, an efficiency gain of about 78%. The platform supports rapid on-site deployment and timely response to changing client requirements, effectively improving software delivery efficiency and success rate.

Keywords: Low-code platform; AI-assisted development; Unified metadata model; Sandbox; Skills; MES

1 INTRODUCTION

In recent years, generative AI represented by large language models has rapidly penetrated every stage of software engineering. From code completion and test generation to producing complete functional modules from natural-language requirements, AI-assisted development tools have greatly reduced coding costs and changed how developers work [1-2]. Controlled experiments show that developers using an AI programming assistant complete a given programming task significantly faster than those who do not [2]. However, the success of an application software project does not depend on coding speed alone.

In enterprise information projects, especially the development and deployment of industry applications such as manufacturing MES and ERP, rework and even project failure are common. Reviews of such projects show that the root cause is often not a technical problem but a discrepancy between the client and the vendor in understanding the business: the client knows the business but struggles to express requirements precisely, while the vendor knows the technology but struggles to grasp business details. Requirements become distorted through repeated translation across documents, meetings and verbal communication, and the deviations surface only during trial operation. Even if AI can generate a functional page within minutes, the coding time saved is offset by later communication and rework if the result does not match the client's mental model of the business logic. Moreover, deployment, configuration and long-term maintenance after delivery also consume substantial labor, and the source-code customization model is particularly cumbersome on sites where requirements change frequently.

Against this background, low-code development platforms, with their visual interfaces and ease of deployment, remain irreplaceably valuable. The concept of low-code was introduced by Forrester in 2014 [3]. Its core idea is to build applications through visual modeling and configuration with as little hand-written code as possible, and to let business staff participate more directly in application design and validation [4-5]. A visual interface lets the client confirm forms, lists and workflows in a “what you see is what you get” manner during development, reducing misunderstanding at the source; a configuration-based delivery approach lets on-site requirement changes take effect without recompiling or republishing source code.

However, existing low-code platforms still face the following technical difficulties in engineering practice: (1) front-end and back-end fields and rules are hard to configure dynamically, since pages, APIs and database structures are usually maintained separately and one change must be synchronized in several places; (2) the security of online scripts is hard to constrain, since the scripting capability opened up for complex business needs may lead to unauthorized access to data and system resources; (3) the reliability of AI generation is hard to guarantee, since large models produce unstable output and the generated code may even introduce security vulnerabilities [6].

To address these difficulties, this paper designs a low-code application development platform that integrates AI-assisted development. Its main contributions are: (1) a unified metadata model that automatically links front-end and back-end configurations; (2) a sandbox mechanism that restricts the data sources, execution permissions and runtime of online

scripts; and (3) purpose-built Skills that constrain the steps and output format of AI generation, making AI-assisted development controllable and reproducible. In addition, the paper validates the platform in a real project through a case study of a manufacturing client's MES. The rest of the paper is organized as follows: Section 2 reviews related work; Section 3 presents the overall architecture; Section 4 describes the key technologies; Section 5 introduces the implementation path; Section 6 presents the application case; Section 7 discusses limitations and future work; and Section 8 concludes.

2 RELATED WORK

2.1 Low-Code Platforms and Model-Driven Engineering

The term low-code was coined in 2014 by Forrester analysts to describe development platforms that deliver applications visually with little hand-written code [3]. Bock and Frank, in a conceptual analysis, note that low-code platforms typically feature graphical modeling, declarative configuration and integrated deployment and runtime, aiming to lower the technical threshold of application development and shorten delivery cycles [4]. Di Ruscio et al. argue that low-code and model-driven engineering (MDE) share the idea of “taking models as the core and generating applications from them”, and can be regarded as two sides of the same idea [5]. Sahay et al. propose a framework for understanding and comparing low-code platforms from the perspective of their features [7]. These studies provide a theoretical basis for the metadata-driven design adopted in this paper, but they pay little attention to security constraints on online scripts and do not discuss integration with AI-assisted development.

2.2 AI-Assisted Software Development

The Codex model proposed by Chen et al. demonstrated, on the HumanEval benchmark, the ability of large models to generate code from function descriptions [1], and tools such as GitHub Copilot subsequently brought this ability into daily development. A controlled experiment by Peng et al. shows that developers using Copilot completed an HTTP-server programming task significantly faster than the control group [2]. Hou et al. systematically reviewed the application of large language models in software engineering, covering requirements, design, coding, testing and maintenance [8]; Wang et al. surveyed autonomous agents based on large language models, which complete complex tasks through planning, memory and tool use [9], offering an idea for making AI complete multi-stage development tasks in prescribed steps. However, Pearce et al. found that code generated by Copilot contains vulnerabilities in some security-relevant scenarios [6], indicating that AI-generated content must be constrained and verified before entering production systems.

2.3 Script Sandboxing and Execution Isolation

Isolating the execution of untrusted or semi-trusted code is a classic problem in system security. The software-based fault isolation (SFI) technique proposed by Wahbe et al. restricts the address space that code can access so that it cannot corrupt the host program [10]. For a low-code platform, online scripts written by implementation engineers or even generated by AI are likewise semi-trusted code, and must be isolated along three dimensions: data sources, permissions and resource consumption.

In summary, existing research has made progress separately on low-code, AI-assisted programming and code isolation, but work that co-designs the three in one platform and lands it in industry-application deployment scenarios remains scarce. This paper addresses precisely this gap.

3 OVERALL ARCHITECTURE OF THE PLATFORM

3.1 Design Principles

The platform follows four design principles. First, metadata-driven: data structures, pages, APIs, validation and permissions are all described by metadata, achieving “define once, consume everywhere”. Second, secure by default: all online scripts and AI-generated artifacts run in a restricted environment, with permissions narrowed by default and opened on demand. Third, visual first, code as fallback: conventional modules are built by drag-and-drop and configuration, while complex business logic is supplemented by scripts or AI-assisted development, and both share the same metadata and permission system. Fourth, human in the loop: AI is responsible for generation and efficiency, while business confirmation, rule review and release decisions remain with people.

3.2 Overall Architecture

The overall architecture of the platform is shown in Figure 1. From top to bottom it consists of the user layer, the design and AI-assist layer, the platform core layer, the framework layer and the data layer, with a permission system and audit mechanism running through all layers to provide cross-cutting protection.

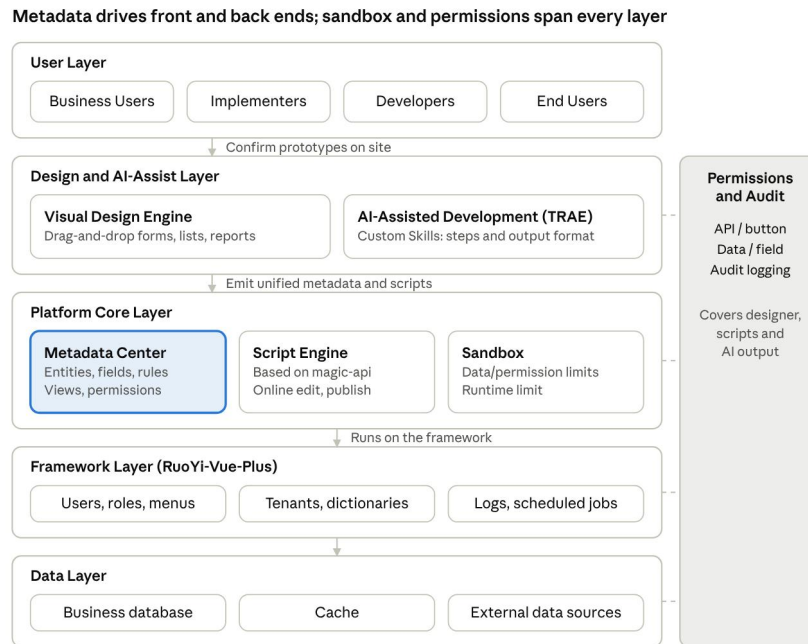


Figure 1 Overall Architecture of the Platform

- (1) User layer. It serves three kinds of roles: clients and business staff take part in confirming forms, lists and reports through the visual interface; implementation engineers complete rapid on-site deployment by drag-and-drop and configuration; developers handle scripts and AI-assisted development for complex business. End users (such as shop-floor operators, quality inspectors and managers) use the published business applications through the web.
- (2) Design and AI-assist layer. It consists of the visual design engine and AI-assisted development. The visual design engine provides drag-and-drop capabilities such as form design, list and query design, report design and page layout; AI-assisted development is based on the TRAE and uses purpose-built Skills to turn business requirements into metadata and scripts that conform to the platform's specifications. The outputs of both enter the platform core layer in the form of unified metadata.
- (3) Platform core layer. This is the key of the platform and consists of three parts: the unified metadata center, the script engine with its sandbox, and the permission and security services. The unified metadata center stores definitions of entities, fields, rules, views and permission points, and provides consistent configuration to the front end and back end; the script engine, based on magic-api, allows APIs and business logic to be written, published and debugged online; the sandbox restricts the data sources, execution permissions and runtime of scripts, and validates and trial-runs AI-generated artifacts.
- (4) Framework layer. Built on an improved RuoYi-Vue-Plus framework, it reuses general back-office capabilities such as users, roles, menus, dictionaries, multi-tenancy and logging, and enhances the permission system on top of them to support API-level, button-level, data-level and even field-level access control.
- (5) Data layer. It includes the business database, cache and external data sources such as device data. For scripts, external data sources can be accessed only through data-source handles registered on the platform; scripts are not allowed to create connections on their own.

3.3 Runtime Workflow

A typical workflow of a business function from requirement to release is as follows. Implementation engineers and the client build a prototype together on site with the designer, and the client immediately confirms fields, pages and workflows; the design result is saved as unified metadata. For complex logic that the designer cannot cover, developers invoke the dedicated Skills, and AI generates metadata and scripts in prescribed steps; the generated artifacts first undergo format validation and a sandbox trial run, and are released after manual review. After release, the platform dynamically generates back-end APIs, front-end pages and permission points from the metadata, with no recompilation or redeployment. This workflow compresses “communication, confirmation, implementation and release” into one visual environment, which is the basis for the platform to reduce business misunderstanding and speed up response.

4 KEY TECHNOLOGIES

4.1 Unified Metadata Model

In traditional development, database table structures, back-end entities and APIs, and front-end forms and lists are maintained separately, so the name, type, length, required flag, dictionary and validation rules of a single field are

defined repeatedly in several places. Once the business changes, they must be modified in several places, which is time-consuming and prone to front-end/back-end inconsistency.

To solve this, the platform establishes a unified metadata model, which can be formalized as a five-tuple $M = (E, F, R, V, P)$. E is the set of entities, corresponding to business objects and their data tables; F is the set of fields, describing the name, type, length, default value, dictionary source and other attributes of each field; R is the set of rules, including validation rules such as required, format and range, and business rules such as field linkage and computed expressions; V is the set of views, describing interface forms such as lists, query conditions, forms and detail pages; P is the set of permission points, describing access control on entities, views and fields. The sets reference one another by identifier: for example, a control in a view references a field, a rule references fields and entities, and a permission point references views and fields.

The unified metadata model brings the following benefits. First, define once, generate everywhere: a single field definition simultaneously drives table-structure changes, back-end create/read/update/delete APIs and parameter validation, front-end form controls and list columns, and the corresponding permission points, so front-end and back-end configurations are linked automatically. Second, single-source validation: a validation rule is defined only once in the metadata; the front end uses it for instant feedback and the back end for final validation, avoiding inconsistent rules on the two sides. Third, traceable changes: metadata carries version information, so additions, modifications and deletions of fields can be traced, and their impact on existing views and scripts can be assessed before release. Fourth, AI-friendliness: structured metadata with a fixed schema is naturally suited as the output target of AI, so that AI-generated content can be machine-validated, which is the basis of the dedicated Skills in Section 4.3.

At runtime, after the platform starts or metadata is published, a metadata loader parses and caches the definitions, and generic CRUD services and front-end rendering components work dynamically from the metadata. This enables dynamic configuration of fields and rules without modifying or redeploying source code.

4.2 Script Engine and Sandbox Mechanism

Pure configuration can hardly cover all business scenarios, such as cross-table statistics, device data processing and complex judgments in work-order state transitions, so online scripting must be opened up. The platform integrates the magic-api script engine, enabling developers to write API scripts online and publish them instantly. However, an online script is essentially semi-trusted code executed on the server. Without constraints, it may read data of other tenants or departments without authorization, invoke dangerous system capabilities, or exhaust server resources through infinite loops or large result sets.

To this end, the platform introduces a sandbox mechanism that constrains scripts along three dimensions: data sources, execution permissions and runtime. Its idea follows classic software isolation techniques [10].

(1) Data-source restriction. A script cannot create database connections or access arbitrary addresses on its own; it can use only data sources registered on the platform, and the visibility of a data source is bound to the current user's role, tenant and data permissions. All database access goes through a platform-provided data-access interface, which automatically appends data-permission conditions, so even if the script author forgets a filter condition, data the user is not entitled to cannot be read.

(2) Execution-permission restriction. The platform applies a whitelist policy to the classes and functions a script may call: reflection, file-system access, system command execution and arbitrary network requests are forbidden by default, and only evaluated utility functions are exposed. The access permission of a script API is inherited from the caller, so the same API returns data of different scope to different roles, and a script cannot elevate its own privileges.

(3) Runtime and resource restriction. A timeout is set for each script execution and the script is interrupted on timeout; the maximum number of rows returned by a single query and the number of concurrent executions are also limited, to prevent an individual script from dragging down the whole service.

In addition, the publication and execution of every script are written to an audit log recording the executor, script version, elapsed time and result status, to facilitate accountability and troubleshooting. The sandbox is also used for trial runs of AI-generated artifacts: before an AI-generated script is formally published, it is first executed once in the restricted environment, and publication is rejected outright if violations such as unauthorized access or timeout occur.

4.3 Purpose-Built Skills

When general-purpose large models generate business code directly, they suffer from unstable output formats, skipped steps and calls to non-existent APIs, and the generated code may also contain security risks [6]. Letting AI “free-wheel” makes the results hard to guarantee. To address this, the platform builds a set of dedicated Skills on ByteDance's TRAE. Here, a Skill can be understood as a reusable instruction package for a specific task, which fixes the background knowledge, operating steps, template examples and checklist of that task so that AI works along a fixed path; in spirit this echoes the planning and tool use of large-model-based agents [9].

Taking the typical Skill “generate a business function module” as an example, its constraints are reflected in three aspects.

First, constraining the generation steps. The Skill splits the task into five steps—requirement clarification, metadata definition, script generation, self-check and import—and requires AI to execute them in order, output the result of each

step upon completion, and ask the developer when requirements are unclear rather than making assumptions. This avoids omissions and makes the process easy to review manually.

Second, constraining the output format. The Skill stipulates that metadata must follow the platform's JSON Schema, scripts must use the functions and data-source handles permitted by the sandbox, and naming must follow the platform's conventions. Built-in templates and examples give AI the “shape of a standard answer”, markedly reducing format deviation.

Third, constraining validation and delivery. The Skill has a built-in checklist requiring AI to self-check after generation whether fields are complete, whether rules conflict, and whether scripts use forbidden capabilities; on import, the platform performs Schema validation and a sandbox trial run, and after these two layers of checks a human still gives the final review.

Through the constraints of Skills, AI changes from a “free-form generating assistant” into an “executor that works by procedure”, and its output becomes metadata and scripts that the platform can machine-validate, thereby improving the reliability of AI generation. Meanwhile, because Skills are preserved as files, the team's implementation experience can be accumulated, reused and iterated, rather than relying on individual developers' prompting skills.

5 IMPLEMENTATION PATH

5.1 Improving the RuoYi-Vue-Plus Framework and Enhancing the Permission System

The platform uses the open-source RuoYi-Vue-Plus framework as its foundation, reusing its mature back-office management capabilities and avoiding duplicated construction of general functions such as users, roles, menus, dictionaries, logging and multi-tenancy. On this basis, to meet the fine-grained permission requirements of industry applications, the permission system has been enhanced: in addition to the original menu and button permissions, API-level permissions, data permissions (by department, by individual, or by custom condition) and field-level permissions (controlling whether a field is visible and editable) have been added. These permission points are linked to the unified metadata center, so that entities, views and fields newly created in the designer automatically generate corresponding permission points, which administrators then authorize uniformly in role management.

5.2 Integrating the Script Engine, Visual Design Engine and Sandbox

For scripting, the magic-api script engine is integrated, and a sandbox interception layer is added at its execution entry to implement the data-source, permission and runtime restrictions described in Section 4.2. For visualization, a drag-and-drop rapid development system has been developed, in which implementation engineers can select controls by dragging, configure fields and query conditions, design reports and preview the effect in real time, as shown in Figure 2. Design results are saved uniformly as metadata and share the same identifiers with the script engine and the permission system.



Figure 2 Page of the Drag-and-Drop Rapid Development System

5.3 Dedicated Skills on TRAE and AI-Assisted Development

For AI-assisted development, dedicated Skills have been built on ByteDance's TRAE around the platform's metadata specification, script specification and sandbox constraints. After a developer describes a business requirement in TRAE, the Skills guide AI step by step through requirement clarification, metadata definition and script generation, and output results that the platform can import directly. For business functions involving multi-table linkage, state transitions and complex calculations, this approach markedly shortens the writing time, as shown in Figure 3.

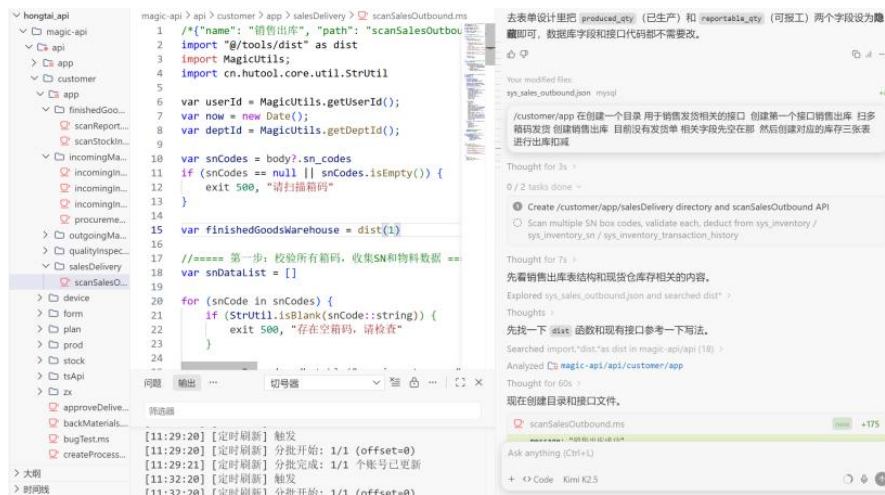


Figure 3 AI-Assisted Development of a Complex Business Function

6 APPLICATION CASE

6.1 Project Background

The platform of this paper was applied when we implemented an MES (Manufacturing Execution System) project for a manufacturing client. Manufacturing sites have numerous and frequently changing business details, and different workshops and positions have different requirements for form fields, inspection items and report definitions, making this a typical scenario for testing the platform's ability to “deploy quickly and respond promptly to changes”. The MES project comprises the following functional modules: basic information, equipment management, mold management, manufacturing management, quality management, system applications, spot-check management, process management, monitoring center and report analysis.

6.2 Implementation Process

During implementation, we used the platform as follows. For conventional modules centered on create/read/update/delete, such as basic information, equipment ledgers, mold ledgers and spot-check items, implementation engineers built them directly on site with the drag-and-drop designer and asked the client to confirm pages and fields on the spot, adjusting immediately when deviations were found, which avoided the long chain of “write documents first, develop next, then accept”. For complex modules involving state transitions and multi-table linkage, such as production work orders and quality inspection plans, developers used the dedicated Skills to have AI generate metadata and scripts step by step, which were released after format validation and a sandbox trial run. For functions such as equipment monitoring that require external data, the data was connected through data sources registered on the platform, with the sandbox restricting the scripts' access scope. The actual interfaces of the platform home page, equipment monitoring, production work-order management and quality inspection plan are shown in Figure 4 to Figure 7.



Figure 4 Platform Home Page

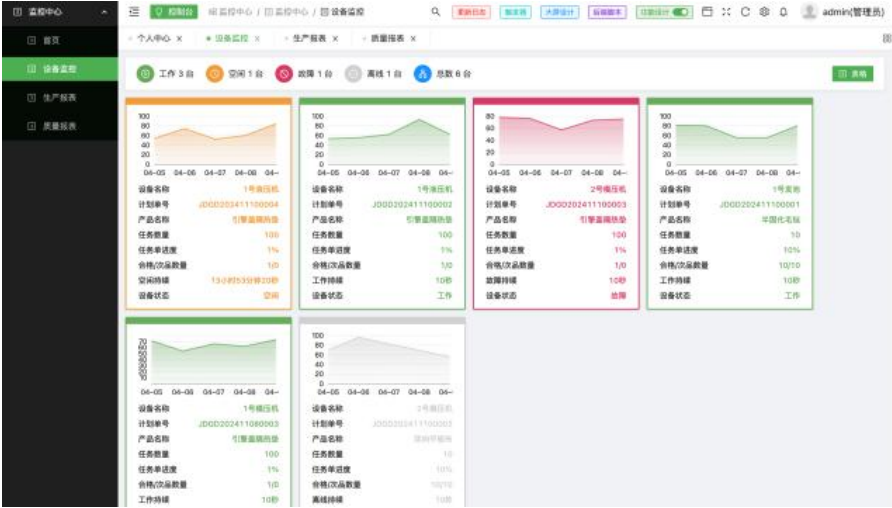


Figure 5 Equipment Monitoring

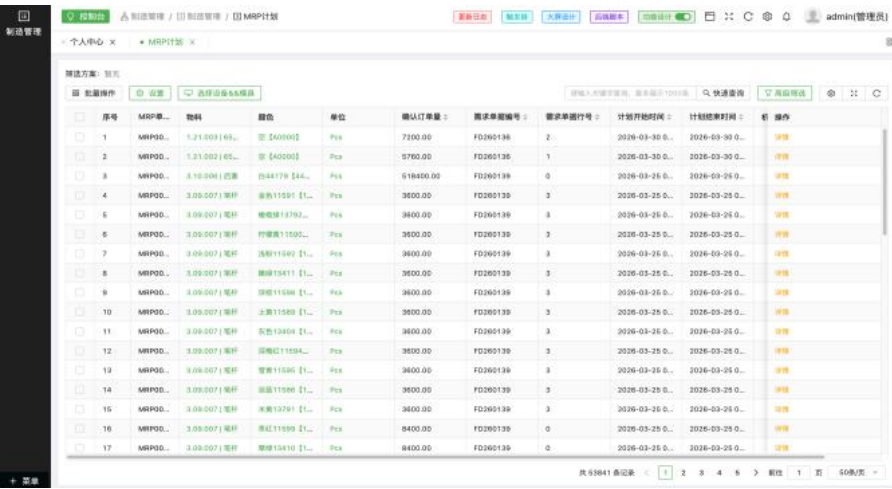


Figure 6 Production Work-Order Management



Figure 7 Quality Inspection Plan

6.3 Results

Compared with the source-code customization model the team used previously, development efficiency based on the platform of this paper improved markedly, as shown in Table 1.

Table 1 Comparison of Development Time for Conventional Modules

Item	Source-code customization	Proposed low-code platform	Improvement
Development time per conventional module	About 1 week	1–2 days	About 78%

Besides the shorter development time, the platform brings three further improvements. First, lower communication cost: the client confirms the business directly through visual prototypes, reducing rework caused by misunderstanding. Second, faster response to changes: adjustments to fields, rules and pages take effect through metadata configuration without recompiling or republishing, and requirement changes raised on site can often be handled the same day. Third, more controllable operation: the sandbox limits the data scope and resource consumption of scripts, the permission system ensures that different positions can access only their own data, and AI-generated artifacts go live only after multiple layers of validation. Therefore, the low-code platform of this paper can help the application software industry deploy rapidly on site and respond promptly to changes in client requirements, effectively improving delivery efficiency and success rate.

7 DISCUSSION AND FUTURE WORK

It should be noted that the efficiency data in this paper come from a comparison of the development time of conventional modules in a single manufacturing MES project; the sample size is limited and the complexity of modules varies, so the generality of the conclusions remains to be verified in more industries and projects. Second, the platform cannot replace all coding work; for functions with complex algorithms or extremely high performance requirements, traditional source-code development combined with platform extension is still needed. Third, the effectiveness of AI-assisted development depends on the capability of the underlying large model and the quality of the Skills, and the Skills themselves must be continuously maintained as the business and platform specifications evolve.

Future work includes: (1) establishing an evaluation mechanism for Skills, collecting metrics such as format compliance, validation pass rate and the amount of manual modification of AI-generated results, to quantify the improvement in reliability; (2) introducing automated generation and execution of test cases to further shorten the verification cycle; (3) extending the unified metadata model to scenarios such as workflow orchestration and report analysis; and (4) accumulating data in more industry projects to conduct a more rigorous controlled evaluation of the platform's efficiency gains.

8 CONCLUSION

To address the business-communication and deployment-efficiency problems that remain prominent in application software projects in the era of AI-assisted development, this paper designs a low-code application development platform that integrates AI-assisted development. The platform links front-end and back-end configurations automatically through a unified metadata model, restricts the data sources, execution permissions and runtime of online scripts through a sandbox mechanism, and constrains the steps and output format of AI generation through purpose-built Skills, thereby addressing the three technical difficulties of low-code platforms in dynamic configuration, script security and AI reliability. Application in a manufacturing client's MES project shows that the platform shortened the development time of conventional modules from one week to 1–2 days, an efficiency gain of about 78%, and can support rapid on-site deployment and timely response to requirement changes. We will continue to improve the evaluation system and verify and promote the platform in more industry scenarios.

COMPETING INTERESTS

The authors have no relevant financial or non-financial interests to disclose.

FUNDING

This work was supported by the 2025 Zhejiang University Students' Scientific and Technological Innovation Project (Project No. 2025R471A012).

REFERENCES

- [1] Chen M, Tworek J, Jun H, et al. Evaluating large language models trained on code. arXiv:2107.03374, 2021.
- [2] Peng S, Kalliamvakou E, Cihon P, et al. The impact of AI on developer productivity: Evidence from GitHub Copilot. arXiv:2302.06590, 2023.
- [3] Richardson C, Rymer J R. New development platforms emerge for customer-facing applications. Cambridge, MA: Forrester Research, 2014.
- [4] Bock A C, Frank U. Low-code platform. *Business & Information Systems Engineering*, 2021, 63(6): 733-740.
- [5] Di Ruscio D, Kolovos D, de Lara J, et al. Low-code development and model-driven engineering: Two sides of the same coin?. *Software and Systems Modeling*, 2022, 21(2): 437-446.
- [6] Pearce H, Ahmad B, Tan B, et al. Asleep at the keyboard? Assessing the security of GitHub Copilot's code contributions//*Proceedings of the 2022 IEEE Symposium on Security and Privacy (S&P)*. IEEE, 2022: 754-768.
- [7] Sahay A, Indamutsa A, Di Ruscio D, et al. Supporting the understanding and comparison of low-code development platforms//*Proceedings of the 46th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. IEEE, 2020: 171-178.

- [8] Hou X, Zhao Y, Liu Y, et al. Large language models for software engineering: A systematic literature review. *ACM Transactions on Software Engineering and Methodology*, 2024, 33(8): 220.
- [9] Wang L, Ma C, Feng X, et al. A survey on large language model based autonomous agents. *Frontiers of Computer Science*, 2024, 18(6): 186345.
- [10] Wahbe R, Lucco S, Anderson T E, et al. Efficient software-based fault isolation//*Proceedings of the 14th ACM Symposium on Operating Systems Principles (SOSP)*. ACM, 1993: 203-216.