Embedded Technology and Intelligent Systems 嵌入式技术与智能系统, 2025, 2(6), 375-390
Published Online December 2025 in Hans. https://www.hanspub.org/journal/etis
https://doi.org/10.12677/etis.2025.26037

基于AI辅助工程的混合MCU－边缘－云平台的工业AIoT系统设计与实现

吴　薇[1,2]

[1]杭州电子科技大学电子信息学院，浙江　杭州
[2]江苏矽望电子科技有限公司，江苏　南京

收稿日期：2026年4月20日；录用日期：2026年5月1日；发布日期：2026年5月27日

摘　要

文章提出一种面向混合微控制器－边缘－云端架构工业AIoT系统的嵌入式AI辅助工程化方法(AI-Assisted Engineering for embedded systems)，用于支持嵌入式软件、RTOS定制、传感器信息模型、通信协议、边缘轻量级学习算法、云端协议及数据挖掘的协同开发，以区别于仅依赖提示生成代码的“氛围编程”(Vibe Coding)。文章以“项目速度”(Project Velocity)原型系统(STM32、RK3588与ThingsBoard)为参考实现，展示了从裸硬件起步、在AI辅助工程化方法下完成端到端软件系统构建的全过程。该架构将信号采集、预处理及故障安全行为部署于微控制器传感器节点，将协议转换、上下文融合、本地推理、数据缓冲、轻量级机器学习及人机交互功能部署于边缘网关，将数据分析、模型注册、受控空中升级(OTA)及运维管理部署于云端。针对嵌入式开发中“AI提示优先”方式易引发的硬件映射缺失、时序约束不清、跨层配置不一致及不安全输出等问题，文章提出面向Zephyr RTOS的“三文件”提示策略，要求AI同时生成硬件覆盖文件(.overlay/.dts)、项目配置文件(prj.conf)和主程序文件(main.c)，从而建立硬件定义、系统配置与应用逻辑之间的闭环。进一步地，文章构建了覆盖需求、实现、验证、文档与部署的规范化工作流程，并结合人工审查、回归测试、硬件在环验证、安全扫描和受控滚动发布等机制，提高系统的可验证性、可维护性与部署可靠性。最后，结合预测性维护用例与多维评价框架，讨论了该方法在实际部署及后续规模化验证中的应用价值。

关键词

工业AIoT，AI辅助工程，边缘计算，Zephyr实时操作系统，负责任AI，系统可追溯性

AI-Assisted Engineering for Developing a Hybrid MCU-Edge-Cloud Industrial AIoT System

Wei Wu[1,2]

[1]School of Electronics and Information Engineering, Hangzhou Dianzi University, Hangzhou Zhejiang
[2]Cynoware Electronics, Inc., Nanjing Jiangsu

文章引用: 吴薇. 基于 AI 辅助工程的混合 MCU-边缘-云平台的工业 AIoT 系统设计与实现[J]. 嵌入式技术与智能系统, 2025, 2(6): 375-390. DOI: 10.12677/etis.2025.26037

Received: April 20, 2026; accepted: May 1, 2026; published: May 27, 2026

Abstract

This paper proposes AI-Assisted Engineering, a structured AI-supported implementation methodology for industrial AIoT systems built on a hybrid microcontroller-edge-cloud architecture. The method supports the coordinated development of embedded software, RTOS customization, sensor information models, communication protocols, lightweight edge learning algorithms, cloud protocols, and data mining, and is intended to address the limitations of prompt-first "vibe coding." Using the Project Velocity prototype system (STM32, RK3588, and ThingsBoard) as a reference implementation, the paper presents an end-to-end software development process that starts from bare hardware and is completed with AI assistance. In the proposed architecture, signal acquisition, preprocessing, and fail-safe behaviors are assigned to microcontroller-based sensor nodes; protocol translation, context fusion, local inference, data buffering, lightweight machine learning, and human-machine interaction are handled by the edge gateway; while data analytics, model registration, controlled over-the-air (OTA) updates, and system management are deployed in the cloud. To mitigate common issues in AI-prompt-driven embedded development—particularly missing hardware mappings, unclear timing constraints, cross-layer inconsistencies, and unsafe outputs—the paper introduces a three-file prompting strategy for Zephyr RTOS, requiring AI-generated deliverables to include a hardware overlay file (.overlay/.dts), a project configuration file (prj.conf), and an application source file (main.c). This strategy establishes a closed loop between hardware definition, system configuration, and application logic. The paper further presents a standardized workflow covering requirements, implementation, verification, documentation, and deployment, reinforced by human review, regression testing, hardware-in-the-loop validation, security scanning, and controlled rolling release. Finally, a predictive maintenance use case and a multi-dimensional evaluation framework are discussed to support practical deployment and future large-scale validation.

Keywords

Industrial AIoT, AI-Assisted Engineering, Edge Computing, Zephyr RTOS, Responsible AI, Traceability

1. 引言

随着工业互联网、边缘计算与嵌入式人工智能技术的持续融合，工业现场系统正由传统的单点感知与数据上传模式，逐步演进为集感知、推理、执行、回传与优化于一体的动态闭环体系[1] [2]。在这一演进过程中，工业系统的智能化不再仅仅体现为云端算法能力的增强，而是越来越依赖于设备侧、边缘侧与云侧之间的协同设计与分层实现。相较于传统以云平台为中心的系统架构，面向工业人工智能物联网(Industrial AIoT, AIoT)的系统构建不仅需要综合考虑算力配置、接口连接与数据流转，还必须系统性地纳入实时响应、故障可恢复、弱网运行、安全可控、版本可治理以及工程可追溯等多重约束。尤其是在设备节点、现场总线、边缘网关、云平台及运维流程深度耦合的复杂应用场景中，系统开发的核心挑战已不再局限于局部算法优化或单板级功能实现，而是演变为贯穿硬件定义、实时软件配置、协议协同、跨层数据一致性、验证测试与部署治理等多个环节的系统工程问题。对于这类混合微控制器–边缘–云端

架构的工业 AIoT 系统，单纯依赖代码生成或模块拼接的开发方式，往往难以满足工程落地对可靠性、可维护性和可验证性的要求。

氛围编程(Vibe Coding)，是指一种以提示驱动为核心、强调根据高层意图快速生成代码的开发方式，但通常缺乏对硬件映射、时序约束、跨层一致性及系统验证的显式处理。面向嵌入式系统的 AI 辅助工程(AI-Assisted Engineering)，是指一种结构化工程方法，其中 AI 在嵌入式系统全生命周期中辅助实现相关工作，但始终受到人工定义的架构、硬件、时序、安全与验证约束，其目标不是生成孤立代码，而是产出可构建、可测试、可部署的系统。

Project Velocity (项目速度)是一个面向嵌入式系统的 AI 辅助工程原型项目，基于 STM32 传感器节点、RK3588 边缘网关和 ThingsBoard 云平台，从裸硬件出发构建面向工业 AIoT 的混合 MCU－边缘－云端可部署系统。

本文的主要贡献如下

(1) 提出一种面向嵌入式系统的 AI 辅助工程方法，将 AI 系统化引入需求分解、硬件映射、RTOS 配置、驱动与协议实现、测试验证、文档整理以及部署维护等开发环节，并在人工主导的架构、时序、安全与验证约束下，形成适用于工业 AIoT 系统的结构化工程流程。

(2) 将面向工业 AIoT 的混合微控制器－边缘－云端系统架构，依据时序敏感性、推理实时性与治理需求进行层次化任务划分：将信号采集与故障安全下沉至 MCU 节点，将协议转换、上下文融合、本地推理与轻量级学习部署于边缘侧，并将数据分析、模型注册、OTA 管理与运维治理部署于云端，同时明确运营技术(OT)与信息技术(IT)的接口边界。

(3) 针对嵌入式项目中 AI 生成代码易出现的硬件映射缺失、配置不一致与逻辑孤立等问题，提出面向 Zephyr 等框架的“三文件”提示策略与上下文注入方法，要求 AI 协同生成硬件覆盖文件(.overlay/.dts)、项目配置文件(prj.conf)和主程序文件(main.c)，从而增强工程项目的可构建性、可调试性与跨层一致性。

(4) 构建覆盖需求、实现、验证、文档与部署全过程的质量保障机制，将人工审查、回归测试、硬件在环验证、安全扫描及受控滚动发布纳入开发流程，并结合预测性维护典型用例与多维评估框架，为后续量化验证与规模化部署提供可操作的方法基础和指标支撑。

2. 氛围编程和面向嵌入式系统的 AI 辅助工程

随着大模型与代码生成技术的发展，基于自然语言提示快速生成程序的开发模式受到广泛关注。其中，所谓“氛围编程(Vibe Coding)”，通常是指一种以提示驱动为核心、强调依据高层意图快速生成代码的开发方式。该方式在原型验证、界面搭建及通用软件场景中具有一定效率优势，但在嵌入式系统开发中，往往缺乏对硬件映射、时序约束、系统配置依赖、跨层一致性与安全验证的显式处理，因而容易产生“代码表面可行而系统整体不可用”的问题。特别是在 RTOS、驱动、中断、总线协议及故障安全机制密切耦合的场景下，这种提示优先、逻辑优先的开发方式存在明显局限。

为应对上述问题，本文提出一种面向嵌入式系统的 AI 辅助工程(AI-Assisted Engineering)方法。该方法并非将 AI 视为替代工程师的自动编码工具，而是将其作为工程协同助手，引入到需求分析、硬件映射、RTOS 配置、驱动与协议实现、测试验证、文档生成及部署维护等开发环节之中，并始终置于人工定义的架构、硬件、时序、安全与验证约束之下。其目标不是生成孤立的代码片段，而是形成可构建、可测试、可部署、可演进的嵌入式系统工程产物。

嵌入式系统中的 AI 辅助工程(AI-Assisted Engineering)，是指在明确工程约束和人工主导验证的前提下，将人工智能作为辅助工具的一种系统化工程方法。其核心目标不是替代工程师，而是提升开发速度、覆盖范围与知识复用效率，同时保证嵌入式系统在时序、资源、接口、安全性、可靠性和可维护性等方面满足工程要求。下面的表 1 是“氛围编程”与面向嵌入式系统的 AI 辅助工程的对比。

Table 1. Comparison between “Vibe Coding” and AI-assisted engineering for embedded systems
表 1. “氛围编程”与面向嵌入式系统的 AI 辅助工程的对比

对比维度	面向嵌入式系统的“氛围编程”	面向嵌入式系统的 AI 辅助工程
主要目标	根据自然语言意图快速生成代码	在真实工程约束下交付可运行、可验证的嵌入式系统
开发方式	提示优先、代码优先	需求优先、架构优先、验证驱动
对硬件的看法	往往被抽象化或默认假设存在	明确建模：引脚、总线、时钟、中断、电源、内存、外设等
AI 的典型输出	孤立代码片段，或“看起来正确”的逻辑	协同交付物：硬件配置、RTOS/项目配置、源代码、测试、文档等
系统上下文感知能力	通常较弱	强调全栈上下文：MCU、RTOS、协议、边缘、云端、部署
对约束的处理	常忽略时序、RAM/Flash 限制、并发、安全状态等	将时序、内存、功耗、安全、接口约束作为一等设计输入
RTOS 集成能力	可能生成任务或逻辑，但缺少有效配置与依赖	能对齐任务、优先级、Kconfig/prj.conf、设备树/overlay、驱动与中间件
软硬件一致性	软硬件一致性	显式保证一致性
验证方式	“先跑跑看”	人工审查、构建检查、回归测试、硬件在环、协议测试、安全检查
输出结果的可靠性	速度快，但在真实产品中稳定性差	前期更严谨，但更适合从原型走向产品
人的角色	事后审查生成结果	担任架构师、约束定义者、验证者和发布责任人
最适用场景	早期创意验证、原型草图、快速实验	真实嵌入式开发、工业物联网、涉及安全的功能、可部署系统

本文以 Project Velocity (项目速度)为原型参考实现，构建了一个基于 STM32 传感器节点、RK3588 边缘网关和 ThingsBoard 云平台的混合 MCU－边缘－云端工业 AIoT 系统[3]，其中轻量级 AI 算法 TinyML [3] [4]处于 STM32 或传统 AI 算法处于 RK3588 中，如图 1 所示。

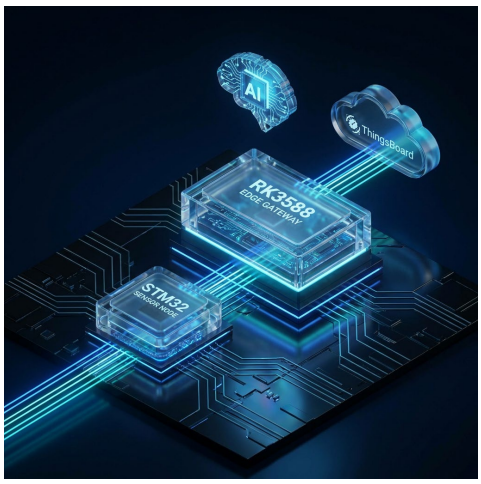

Figure 1. Composition of the “Project Velocity” prototype system
图 1. “项目速度”(Project Velocity)原型系统组成

项目从裸硬件出发、借助本文面向嵌入式系统的 AI 辅助工程(AI-Assisted Engineering)方法完成端到端软件系统构建。在该系统中，信号采集、预处理及故障安全行为部署于微控制器侧；协议转换、上下文融合、本地推理、数据缓冲、轻量级学习算法及人机交互功能部署于边缘侧；数据分析、模型管理、受控空中升级(OTA)与运维治理功能部署于云端。针对嵌入式项目中 AI 生成内容常见的硬件定义缺失、配置文件不匹配、逻辑与工程上下文脱节等问题。

3. 系统需求与总体架构

3.1. 设计目标与分层原则

工业 AIoT 系统设计的核心挑战在于，其各层级在功能、性能与约束上存在根本性差异。底层 MCU 直接对接物理世界的传感器、执行器与现场总线，其设计首要遵循确定性时序、低功耗与故障安全原则；边缘网关承载上下文融合、协议转换、本地推理及现场人机交互，重点关注数据吞吐、缓存能力、断网续传与接口兼容性；而云平台则负责舰队级监控、模型迭代、受控 OTA 升级与审计追溯，强调全局可视、策略一致性与长期治理。因此，系统架构必须在设计初期就清晰划分决策边界——明确“哪些控制回路必须在本地实时闭环内完成，哪些功能可上移至边缘或云端协同”，从而避免因职责模糊导致的时延失控、可靠性下降与运维责任不清。

3.2. 混合 MCU－边缘－云总体架构

“项目速度”采用分层协同的 MCU (STM32)－边缘(RK3588)－云(Thingsboard)混合架构，如图 2 所示。

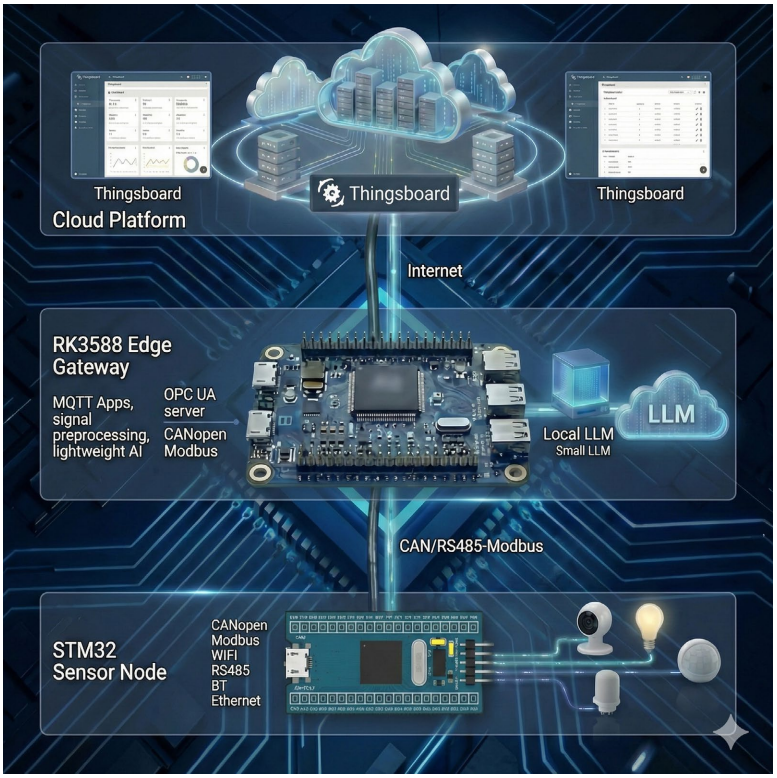

Figure 2. MCU (STM32) - edge (RK3588) - cloud (ThingsBoard) hybrid architecture
图 2. MCU (STM32)－边缘(RK3588)－云(ThingsBoard)混合架构

3.2.1. STM32 传感器/执行节点

在 STM32 传感器/执行节点层，系统主要承担现场感知、状态输入与执行控制等功能。感知与输入部分包括 DHT11/22 温湿度传感器、LM35 温度传感器、热敏电阻分压采样电路、红外接收模块、4 × 4 矩阵键盘、RFID-RC522 射频识别模块以及用于门锁状态检测的磁簧开关等；执行与输出部分包括直流电机驱动模块、继电器控制的电磁锁、I2C 接口 16 × 2 LCD 显示模块以及红外发射电路等。其中，红外发射器与红外接收器共同用于实现对 Lasko 加热器的红外遥控与指令学习功能。节点与上位边缘网关之间通过 UART 串行接口进行通信，并通过 CAN 总线及 CAN 收发器扩展现场总线接入能力；同时，系统还保留了调试与下载接口，用于固件烧录、在线调试与系统维护。整体硬件连接关系如图 3 所示。

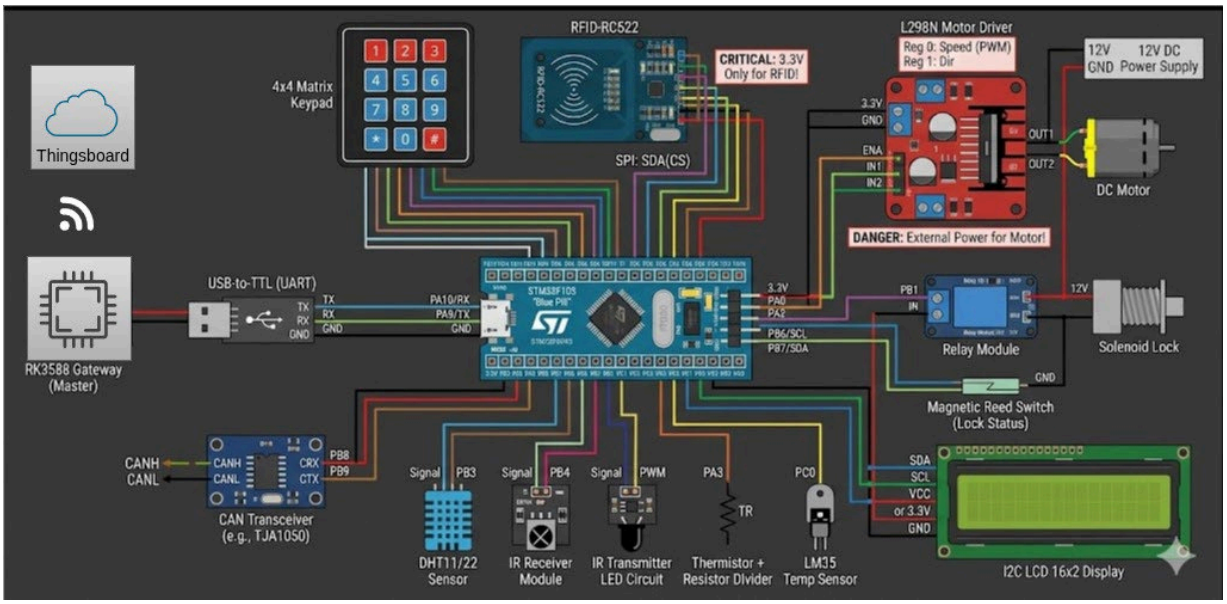

Figure 3. STM32 sensor and actuator node module
图 3. STM32 传感器和执行节点模块

在 STM32 传感器/执行节点的软件平台选择上，本文采用工业级 ZephyrRTOS [5]，而非简单裸机程序或轻量级单任务框架，主要基于实时性、模块化、可扩展性以及后续智能化能力演进等多方面考虑。工业 AIoT 现场节点通常需要同时承担多源传感器采集、执行器控制、周期性任务调度、通信协议处理及异常状态响应等功能，对系统的确定性、并发处理能力和工程可维护性提出了较高要求。ZephyrRTOS 提供线程调度、定时器、中断管理、消息队列、信号量及事件机制，能够较好支持温湿度采集、矩阵键盘扫描、红外收发、LCD 刷新、门锁控制以及与边缘网关的数据通信等任务的协同运行。与此同时，Zephyr 基于 Devicetree、Kconfig 和模块化驱动框架的工程组织方式，可将硬件定义、系统配置与应用逻辑有效解耦，从而提高软件系统的可移植性、可维护性和可测试性，这对于本文所提出的 AI 辅助工程方法尤为关键，因为其能够为 AI 生成结果提供清晰的硬件映射边界和配置约束，减少“逻辑可生成但工程不可构建”的问题。

进一步地，Zephyr RTOS 与 TinyML 在资源受限嵌入式平台上的设计目标具有较高一致性，因此非常适合作为 STM32 节点侧轻量级智能推理的运行基础。TinyML 模型通常强调小参数规模、低功耗、低内存占用和本地实时推理，适合部署在 Cortex-M 等微控制器平台；而 Zephyr 所提供的轻量化内核、可裁剪组件机制、标准化驱动接口与实时任务调度能力，能够为 TinyML 推理任务提供稳定、可控的运行

环境。在实际系统中，模型推理可作为独立线程、定时触发任务或事件响应流程嵌入节点应用，与传感器采样、特征提取、阈值判断和执行控制形成闭环联动，从而实现更低时延、更强鲁棒性的本地智能处理能力。此外，Zephyr 在 UART、I2C、SPI、GPIO、PWM、CAN 等接口上的统一抽象，也便于将 TinyML 与现场感知、通信和控制链路进行集成，降低模型落地的工程复杂度。因而，从实时性、轻量化、组件化配置、外设协同以及本地智能部署能力等方面综合考虑，Zephyr RTOS 不仅是 STM32 工业节点实现的合适基础软件平台，也为 TinyML 在节点侧的工程化部署提供了良好的支撑。

3.2.2. RK3588 边缘网关

在边缘网关层，本文选用 RK3588 作为核心处理平台，主要基于以下考虑。首先，RK3588 属于高性能边缘 AI SoC，集成八核 64 位 CPU 与 6 TOPS NPU，并提供 PCIe、USB 3.0、SATA、双千兆以太网、UART、SPI、I2C 等丰富高速与低速接口，能够同时支撑传统工业协议处理、本地规则引擎、多源数据融合、视频/图像类算法扩展以及边缘侧智能推理等任务，较适合作为工业 AIoT 系统中的边缘计算与协议汇聚节点。其次，RK3588 及其生态已广泛面向机器人、机器视觉、工业控制、边缘计算等场景展开应用，Rockchip 官方资料也明确将 RK3588 视为新一代机器人方案的重要平台之一，这说明其在性能、接口能力与产业生态方面具备较好的工程基础。再次，从软件平台角度看，基于 RK3588 的主流开发板生态已提供对 Ubuntu 22.04 等 Linux 发行版的支持，例如 Ubuntu 22.04 作为长期可运行系统之一，这为长期维护、依赖管理、容器化部署、远程运维以及后续版本演进提供了较成熟的软件基础。对于本文所实现的边缘网关而言，Ubuntu LTS 体系还便于稳定承载 Python、Docker、OPC UA [6]、MQTT [7]、数据库、可视化和 AI 推理相关软件栈，从而降低系统集成与长期维护成本。

基于上述硬件与软件基础，在边缘网关的 RK3588 层，系统运行 Linux/Ubuntu 平台软件栈，实现 OPC UA、CANopen [8]/Modbus 与 MQTT 等异构工业协议之间的实时桥接，并融合多源上下文信息完成本地异常检测、告警生成、数据缓存与规则执行等功能；在云端平台层，则统一承担模型版本管理、远程策略下发、多维可视化分析、系统状态归档以及受控 OTA 更新等职责。该分层设计遵循“越靠近物理层，时序确定性与故障安全要求越高；越靠近系统上层，治理覆盖范围与全局协同能力越强”的原则：MCU 节点负责强实时感知与执行闭环，RK3588 边缘网关负责协议汇聚、局部智能与弱网自治，而云端侧则负责全局分析、模型治理与系统运维。这样的层级划分既发挥了 RK3588 在边缘算力、接口扩展与 Linux 软件生态方面的优势，也有助于提升工业 AIoT 系统的实时性、鲁棒性与可维护性，如图 4 所示。

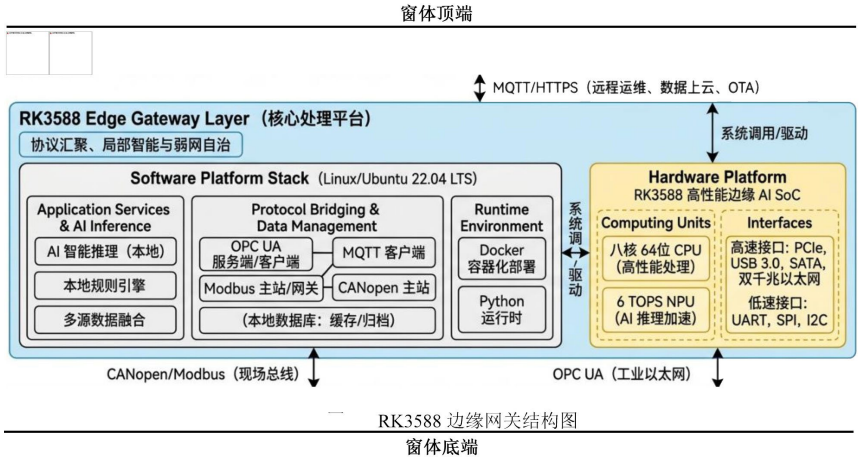

Figure 4. RK3588 edge gateway architecture diagram
图 4. RK3588 边缘网关结构图

3.2.3. 云端 Thingsboard

在云平台层，本文采用 ThingBoard 作为工业 AIoT 系统的统一接入与可视化管理平台，用于实现设备遥测数据接收、远程控制、人机交互展示、规则联动以及历史数据管理等功能。如图 5 所示，系统在 ThingBoard 仪表板中集成了多类现场设备与控制对象的状态监测和远程操作能力，包括 Lasko 加热器红外遥控、直流电机转速与方向控制、LCD 显示内容下发、矩阵键盘输入状态显示、LED 开关控制以及温度等传感器数据的实时可视化。通过这些组件，云平台能够将现场 MCU 节点和边缘网关上传的数据以统一界面进行展示，并支持操作人员通过网页端下发远程控制指令，实现从“状态感知”到“控制执行”的闭环交互。

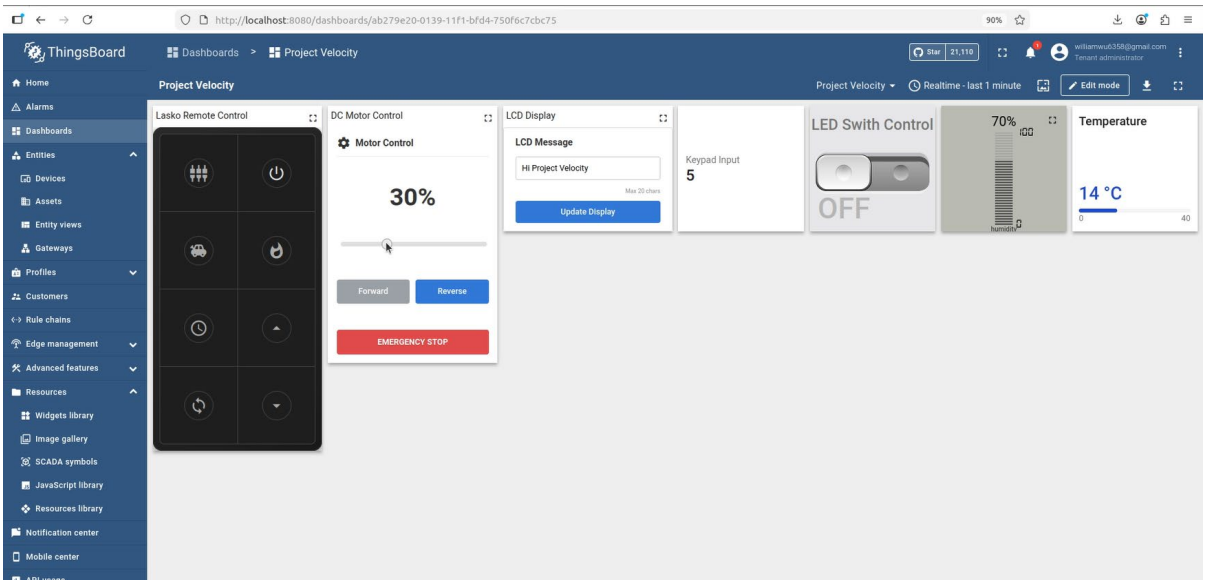

Figure 5. Cloud ThingsBoard dashboard
图 5. 云端 ThingBoard 仪表板

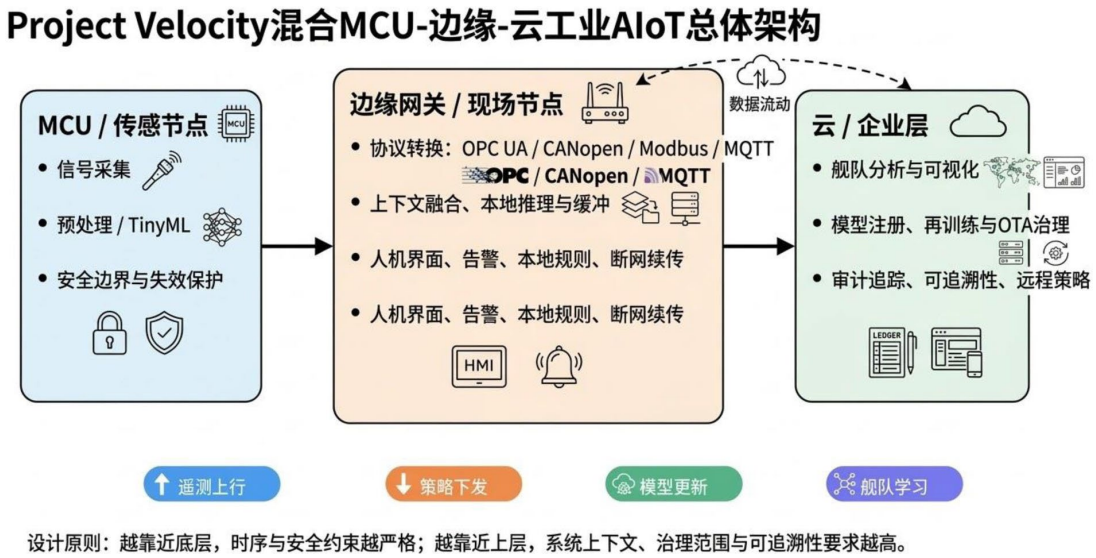

Figure 6. Overall hybrid MCU-Edge-cloud industrial AIoT architecture of “Project Velocity”
图 6. “项目速度”混合 MCU－边缘－云工业 AIoT 总体架构

在系统实现上，ThingBoard 主要承担以下几类功能：其一，作为设备遥测与属性数据的统一入口，接收来自 RK3588 边缘网关经 MQTT 上传的实时数据，并完成存储、展示与历史归档；其二，作为远程控制与运维交互界面，通过 RPC 或命令下发机制，将用户在仪表板上的控制动作转发至边缘网关及 STM32 执行节点，实现电机调速、显示更新、设备开关控制等操作；其三，作为规则与告警管理平台，对温度、状态变化及异常条件进行联动处理，为后续异常告警、日志审计和运维决策提供支撑；其四，作为系统治理与扩展接口，为后续接入模型版本管理、远程策略配置、OTA 升级和多设备集中管理提供统一云端基础。

从架构分层角度看，ThingBoard 云平台并不承担强实时控制闭环，而是侧重于全局可视化、远程访问、历史分析和集中治理。通过将实时性要求较高的感知与执行功能保留在 STM32 节点及 RK3588 边缘层，而将多终端访问、运维管理、可视化分析和策略下发部署于云平台层，系统能够在兼顾现场响应能力的同时，提高整体系统的可管理性、可扩展性和工程部署效率。

3.3. 端到端数据与决策闭环

基于上述分层架构，系统端到端的数据流与决策闭环可抽象为六个相互衔接的阶段：感知、预处理、推理、解释与告警、执行与留痕、再训练与更新。只有当每个阶段之间的交接点均完整保留版本信息、告警依据、现场上下文与执行记录时，系统才具备真正可验证的可信度。换言之，工业 AIoT 系统的关键不仅在于单点识别精度，更在于整个跨层链路在面对真实故障、网络中断与版本迭代时，所保持的全程可追溯性与一致性。

4. AI 辅助工程方法

4.1. 嵌入式与 AIoT 中的“氛围编程”局限性

在简单脚本开发、网页原型构建或纯云端服务场景中，“氛围编程”能够凭借提示驱动快速生成可运行的代码原型，因而具有较高的试错效率和开发速度。然而，在嵌入式系统与工业 AIoT 场景中，其局限性更加明显，且往往直接影响系统的可交付性、可靠性与安全性。

首先，嵌入式开发高度依赖具体硬件平台，AI 生成代码时容易忽略底层硬件的关键约束，例如引脚复用关系、时钟树配置、外设初始化顺序、DMA 通道占用、中断优先级、看门狗策略以及低功耗状态切换等。这些内容通常不是通用代码模板所能自动正确推导的，一旦处理不当，即使代码在语法层面正确，也可能导致系统无法启动、外设异常或现场运行不稳定。

其次，嵌入式与 AIoT 系统通常具有明确的实时性要求。系统是否“基本可运行”并不等同于是否满足工程交付标准。实际应用中，还必须进一步考虑中断延迟、任务调度抖动、资源竞争、故障恢复路径、异常状态下的安全回退机制，以及长期连续运行时的稳定性。因此，“大多数情况下可运行”的代码，在工程实践中往往不足以支撑可维护、可验证、可部署的系统。

再次，AIoT 系统本质上是一个跨层协同系统，其正确性依赖于 MCU、RTOS、驱动、现场总线、边缘网关、遥测接口、OTA 升级机制以及云端数据模型之间的严格一致。无论是字段类型、单位换算、时间戳格式、状态机定义，还是协议报文结构，只要任一层出现错配，就可能引发链路中断、数据解释错误、控制失效，甚至造成远程升级失败和系统级异常。氛围编程通常更关注局部代码生成，而较少显式保证这种跨层一致性。

最后，代码能够编译通过仅表明其在语法和局部依赖层面是成立的，并不能替代真实工程环境下所需的系统验证工作。对于嵌入式与 AIoT 产品，仍必须经过硬件在环测试、协议互操作验证、故障注入测试、长期稳态运行评估、资源边界压力测试以及安全审查等多维验证流程，才能判断其是否真正具备工

程落地条件。换言之，编译成功并不等于系统可用，更不等于系统可信。

综合来看，“氛围编程”在嵌入式与 AIoT 场景中的核心局限，在于其更擅长生成“看起来合理的代码”，却难以天然保证硬件映射正确、时序约束满足、跨层定义一致以及系统验证充分。对于这类强约束、强耦合、强工程属性的系统开发，仅依赖提示驱动的代码生成方式往往是不够的，必须进一步引入面向嵌入式系统的 AI 辅助工程方法，通过架构约束、配置协同、验证流程与人工审查机制，将 AI 的生成能力纳入可控、可验证、可部署的工程框架之中。

4.2. 面向嵌入式系统的 AI 辅助工程

因此，“项目速度”并非将人工智能简单视为孤立的“自动代码生成器”，而是将其纳入一套以工作流驱动的系统工程框架之中，使 AI 参与嵌入式与 AIoT 系统开发的多个关键阶段，并始终受制于工程约束、验证要求与发布治理机制。该方法强调，AI 的作用不在于替代工程师完成最终设计决策，而在于辅助工程师提高需求理解、实现效率、验证覆盖度与文档一致性，从而支撑复杂系统从原型走向可部署工程实现，如表 2 所示。

Table 2. AI-assisted engineering process and guardrail mechanisms
表 2. AI 辅助工程流程与护栏机制

阶段	AI 支持内容	主要产出	约束与护栏
需求	拆解业务用例、定义接口规范、识别危险源	需求说明文档、接口草案、初步风险清单	人工复核系统边界与约束条件
实现	生成固件模板、协议封装代码、API 草稿	驱动框架、服务逻辑、数据模式定义	代码评审 + 静态分析检查
验证	生成单元测试用例、仿真场景、硬件在环(HIL)测试脚本	测试用例集、故障注入脚本、回归测试报告	持续集成(CI)回归测试 + 硬件在环验证
文档	整理设计说明、测试证据、标准运维程序(SOP)	系统设计文档、运维手册、版本发布说明	文档与测试证据留存，并与代码版本绑定
部署	生成监控规则、回滚预案、发布计划	发布实施方案、告警阈值配置、回滚操作方案	灰度发布策略 + 安全扫描审查

在需求分析阶段，AI 可用于辅助拆解业务用例、梳理系统边界、定义接口规范并识别潜在危险源，从而提高需求建模与早期风险识别的完整性。在实现阶段，AI 可加速生成固件模板、驱动初始化框架、协议封装代码、传感器数据模型描述以及边缘与云端接口草案，从而缩短从需求到初始实现的迭代周期。在验证阶段，AI 可进一步辅助生成单元测试用例、异常场景仿真脚本、协议一致性检查逻辑以及硬件在环测试脚本，以提高测试设计的系统性与覆盖范围。在文档阶段，AI 可用于自动整理设计说明、接口文档、测试证据与标准操作流程(SOP)，增强开发成果的可追溯性与知识复用能力。在部署阶段，AI 还可辅助生成发布说明、监控规则、告警策略及回滚预案，为后续运维治理提供结构化支持。

然而，效率的提升不能以牺牲系统可信性为代价。由于嵌入式与工业 AIoT 系统具有实时性强、软硬件耦合紧、现场环境复杂及故障代价高等特点，AI 参与开发流程时必须嵌入明确且强制性的质量护栏。具体而言，应在流程中纳入人工评审、持续集成(CI)回归测试、硬件在环验证、安全扫描以及受控发布机制，以确保 AI 生成内容不仅在语法层面成立，而且能够满足硬件约束、配置一致性、协议互通性与系统安全性的工程要求。

从工程方法论角度看，这种面向嵌入式系统的 AI 辅助工程，与“治理 - 映射 - 测量 - 管理”的风险

控制思想具有一致性，即 AI 不应被视为一次性产出的工具结果，而应被视为需要在系统全生命周期内持续约束、持续验证与持续治理的工程对象。尤其在工业场景中，任何由 AI 生成或辅助形成的设计与实现成果，最终都必须落实为可验证的接口规范、可测试的系统行为、可回退的版本管理机制以及可解释的运维操作指引。只有在这样的前提下，AI 才能真正从“生成代码的工具”转化为“提升嵌入式系统开发能力的工程助手”。

4.3. “三文件”提示策略

为解决 Zephyr 等复杂嵌入式项目中“AI 仅生成应用逻辑代码而忽略底层工程配置”的常见问题，本文提出一种面向嵌入式系统开发的“三文件”提示策略。其核心思想在于：针对任一具体功能需求，在提示设计阶段即明确要求 AI 同步生成.overlay/.dts、prj.conf 与 main.c 三类关键交付文件，从而使生成结果不再停留于孤立的业务逻辑层，而是覆盖硬件描述、系统配置与应用实现三个彼此耦合的工程层次。

其中，.overlay/.dts 文件用于完成硬件描述，主要定义设备树节点、引脚复用关系、总线类型与速率、外设地址及节点标签等内容，是应用程序正确访问底层硬件资源的基础；prj.conf 文件用于完成系统配置，负责显式启用 I2C、SPI、UART、BLE、日志、浮点运算、文件系统等功能所依赖的 Kconfig 选项，从而保证内核能力、驱动依赖与中间件组件在构建阶段被正确纳入；main.c 文件则承载应用层业务逻辑，负责调用系统 API、绑定设备树节点、组织任务流程，并实现状态机、控制逻辑及异常处理路径。三者分别对应硬件定义、软件配置与功能实现，共同构成一个可构建、可运行、可调试的完整嵌入式工程单元。

该策略的工程价值在于，它将抽象的“功能意图”直接映射为嵌入式工程落地所必需的三个关键构成层面，从而避免 AI 仅在逻辑层生成“看起来正确”的代码，却无法在真实工程中完成编译、链接与运行。以 Zephyr 为代表的现代嵌入式框架，其构建过程本质上是应用代码、Kconfig 配置与设备树描述三者的联合收敛过程。若任一环节缺失或不一致，均可能引发驱动未启用、设备节点不存在、头文件依赖缺失、编译选项不匹配，或运行时无法正确寻址外设等问题。因此，三文件同步交付不仅提升了 AI 生成内容的完整性，也显著增强了项目的可构建性与跨层一致性。

此外，该策略还有助于提升调试与问题定位效率。采用三文件同步交付后，一旦在构建或运行过程中出现异常，开发者可以首先判断问题来源究竟属于硬件描述层、系统配置层还是应用逻辑层，并据此进行有针对性的排查。这种分层式的问题定位方式，能够有效缩短调试路径，减少反复修改提示与盲目试错所带来的时间成本，从而提高 AI 辅助开发在复杂嵌入式工程中的实际可用性。

与“三文件”提示策略配套的另一关键机制是“上下文注入”。在提示过程中，必须显式向 AI 提供目标板卡型号、处理器架构、主时钟频率、所使用的传感器类型、总线接口方式、资源限制、实时性要求及安全边界等工程上下文信息，而不能让 AI 仅依据通用经验进行推断。对于深度嵌入式项目而言，越接近硬件层的实现细节，其对上下文完整性和准确性的依赖越强。只有在提示中提供充分、精确且与目标平台一致的上下文约束，才能最大程度减少“逻辑上合理但工程上不可用”的生成结果，并提高 AI 输出与真实系统需求之间的一致性。

总体而言，“三文件”提示策略并非简单增加 AI 的输出数量，而是通过显式约束其交付结构，使 AI 生成结果从“代码片段”提升为“工程单元”，如下图 7 所示。该策略结合上下文注入机制，为 AI 在 Zephyr 等嵌入式框架中的应用提供了一种更具工程可行性的方法路径，也为后续构建可验证、可维护、可部署的嵌入式 AI 辅助开发流程奠定了基础。

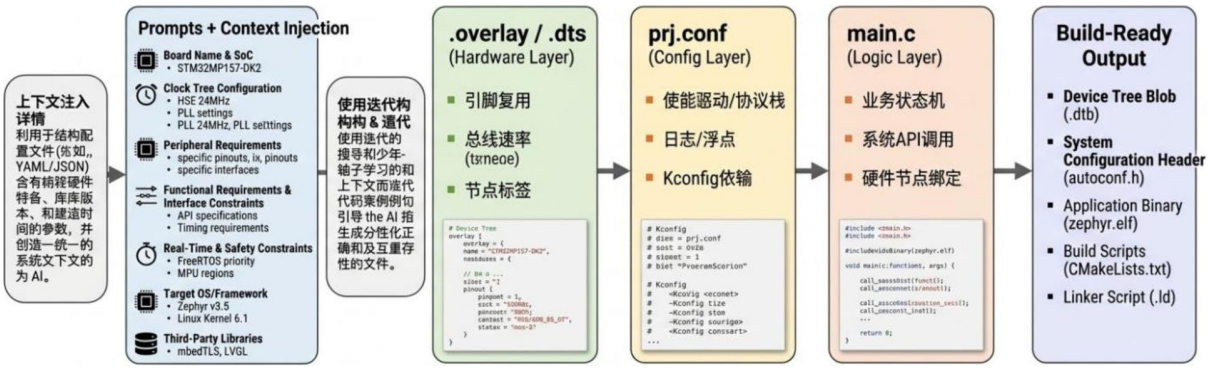

Figure 7. "Three-File" prompting strategy and context injection method
图 7. "三文件"提示策略与上下文注入方法

5. 实现栈与接口设计

在实现层面，系统采用三层架构进行技术栈划分：

MCU 层：主要由传感器驱动、RTOS 任务、信号特征提取模块与失效保护状态机构成。针对需快速本地响应的场景，可在该层执行轻量级阈值判断或 TinyML 推理，以规避总线与网络传输带来的不确定延迟[2] [3]。

边缘层：通常部署具备较强算力与接口扩展能力的工业网关或工控机，负责本地服务容器化、消息缓冲、设备管理、人机界面及多协议桥接。其中，OPC UA 为上层语义互操作提供统一信息建模框架[6]；MQTT 承担轻量遥测与远程指令传递[7]；而 CANopen/Modbus 则用于兼容现场设备与既有工业资产[8]。

云端层：承担对象存储、数据湖、模型注册、舰队看板及 CI/CD + OTA 管理等职能。与传统仅关注业务数据的云平台不同，本项目强调将模型版本、固件版本、配置版本与策略版本置于同一可追溯链条中，从而在发生异常时能清晰回答："当时运行的是哪个模型、哪版配置、在哪些节点上生效过"这一关键工程问题。为此，系统在横向能力上还需集成统一的身份与权限管理、全链路可观测性、策略控制及回滚机制。

从工程实践角度看，真正的难点不在于将单项技术接入系统，而在于确保各层接口在持续版本演进中保持一致性。例如，若现场节点上报的状态字节、边缘层推理结果的置信度字段、云端仪表板的解读信息以及运维人员接收的操作建议缺乏统一定义，将导致模型、软件与运维流程相互脱节。因此，接口规范、版本命名与证据留存机制应与代码一样，被视作系统设计的核心对象予以管理。

6. 负责任 AI 控制与应用示例

工业 AI 系统的构建并非简单地"接入一个模型"即可完成。与办公或消费类应用不同，工业场景尤其强调人在回路、安全覆盖、职责划分与异常处置的确定性。因此，"项目速度"将负责任 AI 的治理要求前置到系统设计阶段，而非事后补充。其核心控制原则包括：安全与人工干预机制、透明性与可解释性、数据最小化与隐私保护、问责与可追溯性，以及鲁棒性与网络安全。

在工程实现上，这些治理原则需要转化为可落地的控制机制，例如在控制链路中引入失效保护状态机与人工接管接口，在告警界面中展示异常原因码及建议检查路径，在更新流程中采用签名包、版本记录与灰度发布策略，并在运维侧保留审计日志、角色访问控制及远程策略变更记录。其核心目标是使 AI 能力始终运行于可验证、可约束、可回退的工程闭环之中。

以"项目速度"为例，该系统由 STM32 传感与执行节点、RK3588 边缘网关和云端 ThingsBoard 平台构成，形成 MCU－边缘－云协同闭环。在该架构中，AI 并不直接越过控制层驱动现场设备，而是通过

边缘策略判断、设备状态校验和末端状态机约束后才可能影响执行动作。例如，在环境调节场景中，即便上层平台发出加热或执行器控制建议，只要本地检测到温度越界、传感器异常、通信中断或人工切换到手动模式，STM32 节点均可拒绝执行该命令并进入预定义安全状态。该机制体现了“AI 可建议、系统可约束、人工可接管”的基本原则。

在透明性与可解释性方面，“项目速度”强调对异常路径和链路状态的显式展示。系统不仅呈现最终运行结果，还可展示关键传感量变化、节点在线状态、边缘转发情况、命令下发时间、执行确认状态及异常原因码。例如，对于“设备无响应”告警，系统可进一步区分为节点离线、总线超时或执行确认缺失等不同故障来源，从而提升故障定位效率，避免将异常简单归因于 AI 本身。

在数据治理方面，系统采用边缘优先处理、按需上云的策略。STM32 节点完成基础感知与局部控制闭环，RK3588 负责协议聚合、数据筛选和局部分析，仅将必要的遥测结果、事件信息和控制状态上传云端。该设计有助于降低带宽与存储压力，也为敏感数据本地处理和隐私保护提供了基础。若未来引入本地视觉模型或本地大模型能力，该分层架构同样支持“数据尽量留在边缘、云端同步结果与元数据”的治理思路。

在问责与可追溯性方面，系统通过完整链路记录实现控制行为的来源可查、过程可追和结果可验。一次控制动作可沿“云端发起 - 边缘转换 - 现场下发 - 节点执行 - 状态回传”的路径进行记录，为故障复盘、责任界定和合规审查提供依据。同时，系统通过本地降级运行、缓存与重传、受控升级与回滚等机制提升整体鲁棒性，避免网络或服务异常直接导致现场功能失控。

需要指出的是，负责任 AI 不仅体现在系统运行阶段，也体现在开发过程中的工程约束机制。“项目速度”所强调的 AI 辅助工程化方法(AI-assisted engineering)，并非让 AI 替代工程过程，而是将 AI 纳入需求分解、接口定义、配置生成、代码实现、测试验证和文档形成等环节，并通过人工审查、回归测试和硬件在环验证等方式对其输出进行约束。由此可见，工业 AIoT 系统中真正关键的并非单一模型性能，而是 AI 是否被置于一个可验证、可控制、可追责、可恢复的工程系统之中。

7. 评价维度与工程验证

由于本文工作的重点在于系统工程方法与组织流程，而非单一算法的精度评估，因此构建了一套涵盖四类维度的综合评价体系：

第一类：系统指标

包括端到端时延、离线可持续运行时间、OTA 回滚耗时等，用于衡量系统在现场环境下的整体可用性与故障恢复能力。

第二类：AI 性能指标

涵盖检测/分类准确率、误报率及模型漂移发现覆盖率，用于评估模型在实际场景中的有效性与稳定性。

第三类：负责任 AI 治理指标

包括告警解释覆盖率、事件可追溯性完整度、策略与权限违规次数等，用于考察系统在透明、可信、可审计方面的治理水平。

第四类：工程效率指标

涉及 AI 辅助节省的开发时间比例、自动生成的测试用例数量、发布后文档与代码的同步更新度等，用以体现 AI 辅助工程在组织与流程层面的实际收益。

该评价体系旨在从系统可靠性、AI 实用性、治理合规性与工程可持续性四个层面，综合反映工业 AIoT 项目在真实部署中的整体成效，如表 3 所示。

Table 3. Evaluation dimensions and representative metrics of Project Velocity
表 3. Project Velocity 评价维度与代表性指标

指标类别	代表性指标	说明
系统 KPI	端到端时延、离线可持续运行时间、OTA 回滚耗时	评估系统在现场环境下的整体可用性、网络中断时的连续运行能力，以及版本故障后的快速恢复能力
AI KPI	检测/分类准确率、误报率、模型漂移发现覆盖率	衡量模型在实际生产场景中的有效性与持续适应能力
负责任 AI KPI	告警解释覆盖率、事件可追溯完整度、权限/策略违规次数	考察系统在输出可解释性、治理闭环建立以及操作审计方面的成熟度与合规性
工程 KPI	AI 辅助节省时间比例、自动生成测试用例数量、文档与代码同步更新度	反映 AI 辅助工程流程对组织开发效率、交付质量一致性以及知识沉淀可持续性的实际贡献

需要强调的是，工业 AIoT 项目不应仅仅追求模型性能指标，而忽视整个工程链路中的综合成本与系统性风险。一个分类准确率更高、但无法实现安全回滚、缺乏结果解释能力或难以跨版本维护的系统，在实际生产环境中往往并不优于一个整体质量更均衡、可管控的解决方案。因此，“项目速度”将“可验证、可审计、可恢复”与传统的“精度、速度、效率”置于同等重要的地位。

“项目速度”目前已完成的是原型系统级验证，而非面向大规模量产场景的长期统计验证。尽管如此，该项目已经完成了若干具有明确工程意义的阶段性验收，表明所提出的方法并非停留在概念层面，而是已经在真实系统链路中得到验证。具体包括以下几个方面：

7.1. 端到端闭环链路验收已完成

系统已完成 STM32 传感节点、RK3588 边缘网关与 ThingsBoard 云平台之间的全链路打通，实现了从现场感知、边缘转发、云端可视化到控制命令下发、末端执行反馈的完整闭环。这表明本文提出的 MCU－边缘－云分层架构具有实际可运行性。

7.2. 协议桥接与跨层协同验收已完成

原型系统已验证现场节点到边缘侧、边缘侧到云端之间的数据转发与协议映射机制，证明异构节点、边缘平台和云端可在统一工程框架下协同工作。这一点对于工业 AIoT 系统尤为关键，因为实际难点往往不在单点模型，而在跨协议、跨设备、跨层级集成。

7.3. 离线退化运行能力已完成原型验证

在云端不可达或链路异常情况下，系统仍可由现场节点和边缘网关维持基本控制逻辑，验证了“边云解耦、现场优先、安全退化”的设计原则。该结果表明系统具备一定的鲁棒性基础，而非完全依赖云端在线运行。

7.4. 可追溯与治理链路已完成原型验证

对控制命令的来源、边缘转发过程、节点执行状态与结果回传路径，系统已具备基本日志记录与事件追踪能力，可支持故障定位和责任追查。这说明本文提出的负责任 AI 治理要求已在工程实现中具有初步落点。

7.5. AI 辅助工程流程已完成可行性验证

“项目速度”采用 AI-assisted engineering 方法完成需求分解、接口定义、配置生成、部分代码实现、

测试辅助和文档整理，显著缩短了从概念到原型系统落地的周期。该结果验证了本文所强调的重点：AI在工业 AIoT 中的价值不仅在于模型推理，更在于提升跨层工程协同效率。

需要强调的是，本文当前提供的验证结果属于原型级、方法级和链路级验收，其目标是证明所提方法“能够工作、能够集成、能够闭环、能够治理”，而非宣称已完成面向大规模商用部署的最终性能定型。因此，本文并未将有限场景下的单次实验结果简单外推为通用结论，而是将其定位为一种经过原型系统验证的工程方法与架构路径。

从工程研究的角度看，工业 AIoT 项目不应仅追求单一模型性能指标，而忽视系统链路中的综合成本与系统性风险。一个即使在局部分类任务上精度更高、但缺乏安全回滚、异常解释、权限约束和版本维护能力的系统，在真实生产环境中并不一定优于一个整体质量更均衡、可验证、可审计、可恢复的解决方案。因此，“项目速度”将“可验证、可审计、可恢复”与传统“精度、速度、效率”置于同等重要的位置。

目前，该方法已在“项目速度”演示系统中完成了整体架构、端到端数据链路、协议桥接、边云协同与工程流程的原型验证，并为后续的硬件在环测试、故障注入测试、长期稳定性测试及更大规模舰队级验证预留了接口。后续研究将重点关注以下方向：

1) 补充更系统的量化实验数据，如端到端时延分布、离线运行持续时间、回滚恢复时间与人工接管成功率；

2) 建立面向设备侧模型漂移的持续监测机制；

3) 完善审计轨迹记录与权限治理策略；

4) 开展跨多场景、长周期的稳定性评估与对比实验。

8. 结论

本文围绕“项目速度”原型系统，提出了一套面向工业 AIoT 复杂系统的 AI 辅助工程方法，系统阐述了包括混合 MCU－边缘－云架构、端到端决策闭环、“三文件”提示策略、负责任 AI 控制机制以及多维评价框架在内的关键技术要素。与单纯依赖对话式代码生成不同，本文强调将 AI 工具整合进受多重质量护栏约束的系统工程流程中，使其在提升开发效率的同时，不破坏嵌入式系统对时序确定性、功能安全与全链路可追溯性的基本要求。

从工程实践角度看，工业 AIoT 系统的关键并非追求某一层的孤立“智能”，而在于实现各层级在职责划分、接口契约、版本治理与故障恢复上的协同一致性。只有当系统能够在本地实现实时、可靠的闭环控制，在边缘侧完成上下文感知与即时响应，在云端实现集中治理与持续学习，并且整个链路始终保持可解释、可回滚与可审计，AI 辅助工程才真正具备在工业场景中规模化落地的价值。

需要说明的是，“项目速度”的作用在于完成原型级系统验证与链路验收，而非形成某一单任务上的算法竞赛结果。后续我们将结合“项目速度”，作为一个基于 AI 辅助工程的混合 MCU－边缘－云平台的工业 AIoT 系统的开源项目，详细描述硬件的实现，边缘 RK3588 的 AI 算法的实现，本地 LLM 的集成，以及各个行业的应用，包括具身智能。大家可以通过[9]获取最新的进展。

参考文献

[1] Shi, W., Cao, J., Zhang, Q., Li, Y. and Xu, L. (2016) Edge Computing: Vision and Challenges. *IEEE Internet of Things Journal*, **3**, 637-646. https://doi.org/10.1109/jiot.2016.2579198

[2] 吴薇, 吕亚欣. 服务国外市场的 AIoT 开源方案及其应用[J]. 单片机与嵌入式系统应用, 2022, 22(9): 4-9.

[3] Warden, P. and Situnayake, D. (2020) TinyML: Machine Learning with TensorFlow Lite on Arduino and Ultra-Low-Power Microcontrollers. O’Reilly Media.

[4] Lin, J., Chen, W.-M., Lin, Y., *et al*. (2020) MCUNet: Tiny Deep Learning on IoT Devices. arXiv:2007.10319.
[5] The Zephyr Project (2026) Zephyr Project Documentation. https://docs.zephyrproject.org/latest/index.html
[6] OPC Foundation (2026) OPC Unified Architecture (OPC UA), IEC 62541 Overview.
[7] OASIS Open (2019) MQTT Version 5.0.
[8] CAN in Automation (CiA) (n.d.) CANopen CC—The Standardized Embedded Network.
[9] http://www.cynoware.com/cynoware/