DOI:10.19651/j.cnki.emt.2520110

基于规范解析的 AUTOSAR 测试数据自动生成*

王 辉[1,2] 李超超[2] 汪济州[2] 徐封杰[2] 方 菱[2]
(1.安徽大学物质科学与信息技术研究院 合肥 230000; 2.中国科学院合肥物质科学研究院 合肥 230031)

摘 要: 汽车开放系统架构(AUTOSAR)标准是以文档形式呈现,规模庞大且有模糊性。验证车载软件的一致性测试需从文档中提取出配置,以保障车载软件合规性。在深入研究 AUTOSAR 规范文档的内容及一致性测试配置的前提下,提出了一种自动化生成配置数据的方法。首先,在图检索增强生成技术的基础之上集成 MinerU,利用语义增强、文本优化、结构感知分块策略,构成领域实体-关系知识网络。其次,采用少样本思维链(Few-Shot CoT)提示词技术,引导大语言模型(LLM)从领域知识网络中提取测试场景及所需配置数据。最后,基于配置规范生成带属性的配置项树,通过广度优先搜索(BFS)和深度优先搜索(DFS)算法检查并补全配置数据,生成符合 AUTOSAR 标准的 ARXML 配置文件。在 15 个 AUTOSAR 基础软件模块实验中的 8 项指标上,对比传统生成配置数据的方法,结果显示所提方法在隐式依赖提取的 F_1 分数达到 97.51%,人工修正率降低至 5%,测试场景生成数量提升 3%。用本研究提出的方法生成的配置,不仅可以协助测试人员发现软件中难以定位的异常,而且显著提升隐式配置依赖项提取的完整性与配置文件生成的质量和效率。

关键词: AUTOSAR 规范文档;配置数据自动生成;规范解析;一致性测试
中图分类号: TN409 **文献标识码**: A **国家标准学科分类代码**: 510.1050

Automatic generation of AUTOSAR test data based on specification

Wang Hui[1,2] Li Chaochao[2] Wang Jizhou[2] Xu Fengjie[2] Fang Ling[2]
(1. Institutes of Physical Science and Information Technology, Anhui University, Hefei 230000, China;
2. Hefei Institutes of Physical Science, Chinese Academy of Sciences, Hefei 230031, China)

Abstract: The automotive open system architecture (AUTOSAR) standard is provided in the form of large-scale textual specifications that are extensive and inherently ambiguous. To verify the compliance of in-vehicle software, consistency testing requires the extraction of configuration information from these documents to ensure conformity with the standard. Based on an in-depth analysis of AUTOSAR specifications and consistency test configurations, this paper proposes an automated method for generating configuration data. First, MinerU is integrated into a graph-based retrieval-augmented generation framework to construct a domain-specific entity-relation knowledge network through semantic enhancement, text refinement and structure-aware chunking strategies. Second, a few-shot chain-of-thought (Few-Shot CoT) prompting technique is employed to guide a large language model (LLM) in extracting test scenarios and the required configuration data from the domain knowledge network. Finally, a configuration item tree with attributes is generated according to configuration rules, and breadth-first search (BFS) and depth-first search (DFS) algorithms are applied to validate and complete the configuration data, resulting in ARXML configuration files compliant with the AUTOSAR standard. Experimental results on 15 AUTOSAR basic software modules, evaluated using 8 metrics and compared with traditional configuration data generation methods, show that the proposed approach achieves an F_1 score of 97.51% in implicit dependency extraction, reduces the manual correction rate to 5% and increases the number of generated test scenarios by 3%. The configurations generated by the proposed method not only assist testers in identifying hard-to-locate software anomalies but also significantly improve the completeness of implicit configuration dependency extraction as well as the quality and efficiency of configuration file generation.

Keywords: AUTOSAR specification documents;configuration data automatic generation;rule analysis;conformance testing

0 引 言

随着汽车电动化、智能化及网联化的迅猛发展,为应对汽车电子控制单元(electronic control unit,ECU)数量激增所带来的软硬件高度耦合、复用性差、兼容性问题凸显以及系统集成复杂度急剧上升等挑战,汽车开放系统架构标准(automotive

收稿日期:2025-10-25
* 基金项目:国家自然科学基金(32427801)项目资助

open system architecture,AUTOSAR)应运而生[1-2]。

AUTOSAR 标准中的软件规范文档旨在对单个基础软件模块(basic software,BSW)或运行时环境(runtime environment,RTE)的行为、标准接口、配置参数以及模块间交互等方面进行相应约束。一致性测试作为一种黑盒测试,主要通过在不同场景下对比多样化输入对应的输出结果与预期结果的一致性,来验证软件模块是否符合相应的 AUTOSAR 规范[3-4]。由于 AUTOSAR 规范文档主要采用自然语言描述而存在模糊性。不同供应商依据此文档开发的软件需通过一致性测试,以验证其合规性[5-6]。

在一致性测试过程中,测试数据通常是由多个配置项及其取值组合而成的配置数据,其生成过程要求测试人员深入理解规范文档,并据此手动创建配置数据。这种传统的配置数据生成方式不仅耗时巨大,且易出错[7]。这是因为每个测试场景的配置数据并非仅由规范文档中的单条规范所决定。单条规范往往未能提供完整的约束条件、上下文关联信息及隐式依赖关系,从而导致语义不完整,必须依赖测试人员对规范文档上下文的深入解读与分析。此外,被测模块所涉及的配置项数量众多,且这些配置项之间的嵌套与约束关系错综复杂,不同功能测试所需的配置项及其取值也不尽相同。配置数据直接影响被测模块在测试场景下的运行行为,不同配置组合能实现测试场景的多样性,从而为车载软件的功能正确性和可靠性提供重要保障。由此可知,高效且高质量的自动化配置生成不仅能显著提升一致性测试的开发效率,减少人工错误,还能系统性地覆盖更多规范要求下的测试路径,增强测试的全面性和鲁棒性。

在自动化测试数据生成领域,相关研究人员进行了一定的研究。Wang 等[8]提出了用例建模与测试生成(use case modeling and test generation,UMTG)方法,该方法通过受限用例建模(restricted use case modeling,RUCM)对用例描述进行结构化限定,并结合多种自然语言处理(natural language processing,NLP)技术(包括语义角色标注、词性分析)自动提取系统行为、条件与状态变化信息。基于这些解析结果,UMTG 将自然语言条件自动转换为对象约束语言(object constraint language,OCL),并利用 Alloy 求解器生成满足不同路径条件的测试数据。尽管 UMTG 显著提升了测试数据的自动化程度,但其仍存在一些限制:首先其依赖 RUCM 模板化写法,难以直接处理自由形式的自然语言规范;其次它对领域模型质量依赖较强,模型不一致会降低约束推导的准确性;此外,面对复杂语义时它仍需手动补写 OCL。

臧远山[9]提出了一种基于知识图谱的 CTCS-3 车载设备测试数据自动生成方法。该方法从 CTCS-3 规范及相关技术文档中抽取设备、功能、消息及其属性与约束关系,构建领域知识图谱,以统一表达测试相关的结构化知识。在此基础上,方法通过遍历知识图谱中的实体及约束关系自动推导测试项,并利用基于规则的约束推理算法生成满足输入参数类型、取值范围及依赖关系的测试数据,减少了对人工解析规范文档的依赖。然而,该方法知识图谱的构建高度依赖领域专家进行实体与规则抽取、搭建和维护成本较高,且容易产生遗漏。Wang 等[10]提出的 SIT-SE 方法将形式化规格说明 SOFL、动态符号执行以及约束求解技术结合,以实现路径级别的自动化测试数据生成。该方法首先将 SOFL 规格分解为互斥的"功能场景",并通过符号执行提取每条执行路径的路径条件与符号终态,进而构造"路径定理"以验证该路径与功能场景之间的语义一致性。随后,SIT-SE 通过分支序列覆盖 BSC 策略对符号执行进行增量扩展,由 SMT 求解器自动求解满足路径条件的新输入,从而逐步生成可覆盖不同控制流序列的测试数据。尽管该方法具有较高的自动化程度,但其应用仍强依赖完备的形式化规格说明且复杂情况下 SMT 求解会失败。Shigihara 等[11]提出一种面向 AUTOSAR 操作系统(operating system,OS)模块的测试程序自动生成框架,通过使用组合测试工具 PictMaster 生成大量应用程序编程接口(application programming interface,API)参数组合的测试空间,然后将其转换为 YAML 格式的 TESRY 抽象场景描述,再由其开发的 AKTG 工具自动生成 ARXML 配置文件。该方法的局限性在于 TESRY 本质上是一种描述语言而非自动推导机制,测试数据的生成仍需依赖人工从 AUTOSAR OS 规范进行场景设计,并没有完全实现自动化。Ei-Gnainy 等[12]提出了一种基于 AI 的 AUTOSAR 配置数据自动化生成方法,该方法构建了一套从自然语言描述的规范到标准化 ARXML 配置文件的端到端生成流程。通过构建英文规范与 YAML 结构化配置之间的小规模映射数据集,并对 LLaMA2 进行微调,使模型能够根据自然语言生成对应的 YAML 配置;随后再使用 Python 脚本将 YAML 转换成标准 ARXML 配置数据文件。这样可将"语义理解"与"结构生成"分离,降低了大模型直接生成 ARXML 的难度,从而使自动化程度显著提升。然而,该方法的训练数据需要人工构建且仅覆盖单一模块 CanNM(CAN network manager),导致方法缺乏跨模块的通用性。其次,模型仅学习规范到 YAML 的静态映射,无法理解规范之间的依赖关系。

上述各项研究表明,现有方法在处理自然语言描述的规范时,所采用的自动化技术生成的测试数据仍存在显著局限性。为克服现有方法的局限,本文在确保测试数据生成质量的基础上,提出了一种创新的自动化测试数据生成方法。该方法首先借鉴知识图谱方法的"结构化表示"思想,但不再依赖人工构建规则,而是引入结合了 MinerU 的图检索增强生成技术,实现对规范文档的动态解析和实体/关系的自动抽取,从而构建可扩展的领域知识库,增强模型对复杂语义的理解能力。其次,针对具有关联关系的规范,在自动化处理其组合推理难题时,本文利用 Few-Shot CoT 提示策略引导大语言模型(large language model,LLM)在领域知识库中实现多规范联合推理,以自动提取更多测试

场景及其所需的测试数据，从源头减少人工设计依赖及遗漏。此外，面对测试数据中配置项之间存在的隐式依赖和嵌套结构，本文设计了带属性的配置项树模型与数据生成算法自动完善必要的隐式配置项及其数值，生成标准化的 ARXML 配置数据文件。最终的实验结果表明，本文方法有效综合了领域知识表示、语言模型推理以及结构化数据生成的优势，在自动化程度、规范理解精度、配置项推导覆盖范围及 ARXML 文件生成质量方面均显著优于现有方法，为 AUTOSAR 一致性测试的配置数据自动生成提供了一种更高效、更通用且更可靠的解决方案。

1　AUTOSAR 软件规范文档

AUTOSAR 软件规范文档是生成一致性测试数据的核心依据，其文件命名遵循 AUTOSAR_SWS_{MODULENAME} 格式，其中 MODULENAME 代表具体功能模块的名称（如 OS 等）。文档内容主要以自然语言描述，涵盖 3 大核心组成部分：功能规范（functional specification）、应用程序接口规范（API specification）和配置规范（configuration specification）。这 3 部分共同界定了模块的功能行为、接口要求及配置参数等相关规则[13]。

1.1　功能规范与 API 规范

对于功能规范和 API 规范，每条规范均通过唯一的 SWS-ID 号进行标识，如表 1 所示。SWS-ID 号为规范条目提供了全局索引，便于跨文档的引用与关联分析。功能规范主要用于描述模块的核心功能逻辑，包括任务调度规则和事件触发条件等，例如 SWS_Os_00002 规范规定了 OS 模块需按照偏移量递增的顺序处理调度表中的所有到期点。API 规范主要定义模块接口的行为约束，包括服务函数的输入输出参数及其执行逻辑，例如 SWS_Os_00358 规范规定了 StartScheduleTableAbs 服务的绝对启动机制[13]。

表 1　自然语言描述的规范

Table 1　Natural language descriptions of specifications

SWS-ID 号	描述内容	功能规范/API 规范
SWS_Os_00002	The operating system module shall process each expiry point on a schedule table from the initial expiry point to the final expiry point in order of increasing offset.	功能规范
SWS_Os_00411	The operating system module shall make use of ticks so that one tick on the counter corresponds to one tick on the schedule table.	功能规范
SWS_Os_00407	An expiry point shall activate at least one task OR set at least one event.	功能规范
SWS_Os_00358	StartScheduleTableAbs service starts the processing of a schedule table at an absolute value “Start” on the underlying counter.	API 规范
SWS_Os_00399	IncrementCounter service increments a software counter.	API 规范
⋮	⋮	⋮

1.2　配置规范

配置规范详细定义了构成配置数据的各项配置项的全面介绍，涵盖模块（Module）、容器（Container）和参数（Parameter）3 大类别及其层级结构与相互间的依赖关系。如图 1 所示，配置项按照模块→容器→子容器/参数的层级结构逐级嵌套，容器始终作为参数或子容器的上级节点。所有配置项（包括容器和参数）的存在性及其实例数量的限制均由多重性（Multipl-icity）属性进行约束。此外，参数还特别包含布尔型（Boolean）、整型（Integer）等构成的类型（Type）及取值范围（Range）的配置属性。对于某些特定容器（如 Reference 引用容器），还需通过目标（Target）属性明确指定其引用的目标容器名称。这些要素共同确保了配置项的完整性与关系追溯的准确性。

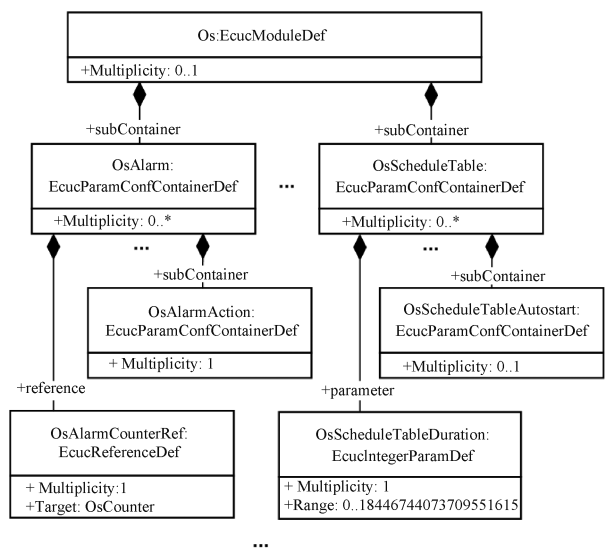

图 1　配置项层级结构及属性约束示意

Fig. 1　Schematic of configuration item hierarchy and attribute constraints

尽管文档中的一条条规范是独立存在的，但在实际应用中，受自然语言描述规范存在一定模糊性的影响，通常需要将多条规范组合起来才能推导出完整、无歧义的测试目的及所需配置项。以 SWS_Os_00407 为例，它规定了“到期点必须激活任务或事件”，却未说明“谁的到期点”、

"到期点到期的判定标准"等关键前提。它显式声明了"任务或事件"配置项,却未明确指出与之关联的必要配置项(隐式配置项),这些隐含信息只能通过与其他规范联合分析才能明确,否则会因描述片段化而生成残缺或错误的测试目的和配置数据。此外,针对长下文理解受限问题,需将文档内容进行整合后再分析,充分把握所有信息之间的结构和关联性。

由于上述技术的局限性,传统方法难以高效解析具有模糊性的自然语言规范,并完整提取隐式依赖。因此,本文提出一种融合图检索增强生成技术的创新框架,将预训练的参数化记忆(如 LLM)与非参数化记忆(如规范文档等外部知识)相结合[14-15]。在自动构建结构化的外部知识库后,借助 LLM 的上下文理解能力与外部知识库的检索机制,实现模糊规范的动态解析及隐式依赖的自动化提取。

进一步地,为更好地表示从上述框架生成的测试场景下各配置项之间的嵌套关系,本文采用 ARXML 文件格式存储最终生成的配置数据。ARXML 是一种用于描述 AUTOSAR 内部数据和信息的可扩展标记语言(extensible markup language, XML)格式[16-17]。

2　自动化配置数据生成方法

在 AUTOSAR 中,一致性测试体系由 4 个部分组成:源代码检查、配置检查、操作签名测试以及动态测试(前 3 部分可统称为静态测试)。源代码检查主要通过静态分析的方法,检查 AUTOSAR 各模块的源代码文件(通常是 C 头文件)在实现过程中是否包含符合规范要求的操作原型(operation signatures)和类型声明(type declarations)。配置检查则用于验证配置文件中的各项配置(不同的配置可以使被测模块通过具有不同接口和行为而产生不同的变体)是否遵循规范要求。操作签名测试旨在检查被测模块是否按要求提供或需要相应的 API。而动态测试则验证被测系统(system under test,SUT)在动态运行时的行为(需依赖配置文件中的配置数据来驱动)是否符合规范要求。

为了提高一致性测试的质量和效率,本文研究的基于知识整合的配置数据自动生成方法被划分为两个阶段进行,如图 2 所示。第 1 阶段涉及测试场景及所需配置项的生成;第 2 阶段则涵盖配置项的检查、补全以及完整配置数据文件的生成。这两个阶段相互补充,第 1 阶段主要通过图检索增强生成框架解析自然语言描述的规范文档来实现,而第 2 阶段则基于第 1 阶段的生成结果,并结合数据生成算法,最终产出配置数据文件。

2.1　构建实体-关系知识库

首先,鉴于规范文档中的标题、图片、图表及格式在传达上下文信息方面具有不可或缺的作用,本文引入开源服务 MinerU 作为文件加载器[18]。该服务不仅有利于后续维护,还能在不增加原有框架冗余的前提下,用于预处理

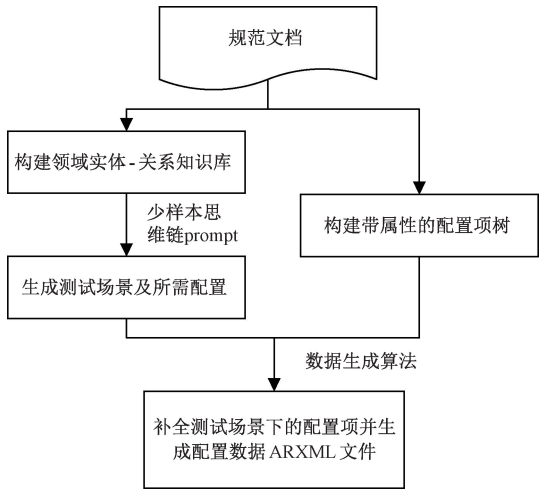

图 2　自动化配置数据生成方法架构

Fig. 2　Architecture of the automated configuration data generation method

包括 PDF 格式文件在内的输入规范文档。通过对输入进行局部分析、公式检测与识别、表格处理以及 OCR 识别等一系列内容解析,最终生成保留这些信息的 Markdown 格式结构化文档。采用这种提取方法,不仅完整保留了文档的格式和内容,也便于大模型进行后续处理。

然而,对于表格和图片这类数据,仍需对提取出的内容进行语义增强和文本优化。具体而言,通过调用大型模型接口对表格内容进行总结和摘要;在获取图片的前后文作为背景信息后,利用视觉模型生成图片的详细描述。最终,将这些信息以注释形式嵌入 Markdown 文档中,从而获得经过预处理的文本。

其次,由于不同的 LLM 在处理文档时均受其最大上下文长度的限制,因此需要文档切分器。传统的检索增强生成(retrieval-augmented generation,RAG)通常采用固定大小的分块切分方法,即基于 token 的策略进行切分,这种方式容易破坏上下文的连续性[19]。本文方法则采用结构感知分块策略,即先按照标题层级划分块,确保标题下的所有内容在同一块中,然后对文本块使用滑动窗口进行分割,并允许窗口间存在重叠;对于表格和图片块则不进行分割,单独成块,直接关联其描述和上下文。

接着,对划分后的每个块,利用 LLM 结合领域特定的提示词模板,提取规范中涉及的实体(如模块、容器、参数、规范条目)及其关联关系,并保留实体在自然语言中的详细描述。在此基础上,通过将同一实体的不同表述形式(例如"OsScheduleTable"与"schedule table")进行统一映射,构建初始的领域实体图。为进一步增强知识的可检索性,引入大型语言模型对实体图中的节点及边进行语义总结,例如将"SWS_Os_00002"规范描述的调度表处理逻辑转化为简洁的语义标签。随后,采用 Leiden 算法[14]对实体图进行社区检测,通过聚类分析将语义关联紧密的实体

归入同一社区(如“处理调度表”相关实体群组),并利用 LLM 为每个社区生成摘要性描述,以概括各社区的主要内容。

通过以上一系列操作步骤,不仅成功将原始规范文档构建为分层次、动态扩展的领域实体-关系知识库,还为后续大语言模型从复杂的外部文档中进行高效检索提供了便利。该过程的整体流程如图 3 所示。

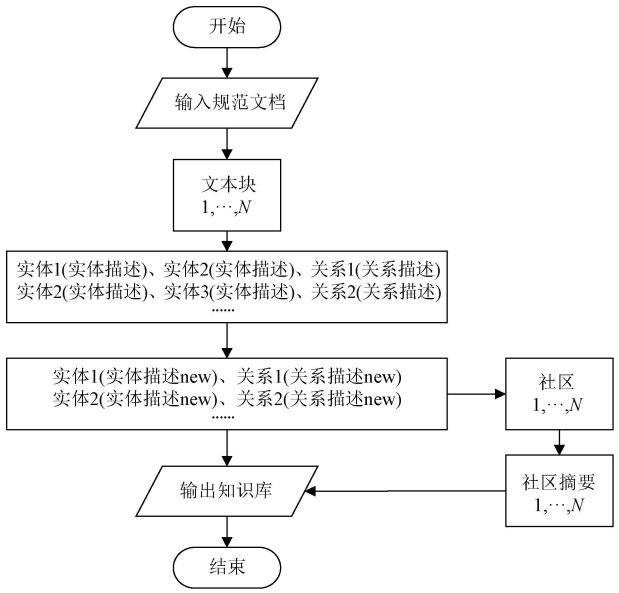

图 3　构建领域实体-关系知识库流程

Fig. 3　Flowchart for constructing the domain entity-relationship knowledge base

2.2　测试目的及对应配置项生成

在基于规范的测试中,明确测试目的是高效生成配置数据的前提条件,只有这样才能准确识别每个测试目的对应的测试场景下所需的配置项(如容器、参数等),进而构建符合 AUTOSAR 规范的配置数据。这一过程的逻辑闭环体现为:测试目的需从单个或多个规范的语义关联中推导,而配置项的生成则依赖于测试目的下对应测试场景的具体需求。配置项由显式声明的配置项及相关的隐式配置项组成,前者通过解析规范得到,后者则在前者基础之上由构建的知识库推导得出。例如,综合 SWS_Os_00002(调度表处理顺序)、SWS_Os_00411(计数器和调度表的同步关系)、SWS_Os_00407(到期点激活任务或事件)3 条规范,可得出测试目的“验证调度表按到期点偏移量等于计数器值的顺序执行任务激活”,进而从中提取出 OsScheduleTable 容器、ExpiryPointOffset 参数等显式配置项。

然而,鉴于 AUTOSAR 规范文档的复杂性,直接从长文本中完成逻辑推理与配置项提取面临显著挑战。为此,本研究引入 Few-Shot CoT 技术,该技术融合了小样本学习(few-shot learning)与思维链(chain-of-thought, CoT)的核心思想[20-21]。一方面,通过提供少量标注样本(如 3～5 个典型测试目的与配置项的生成案例),引导 LLM 快速适应领域任务,避免模型微调带来的高成本;另一方面,采用分步推理策略,将测试目的生成拆解为“规范检索→逻辑关联→目的提炼”的递进过程,有效缓解 LLM 在处理长文本时可能出现的逻辑断层问题。

如图 4 所示,在提示词设计中,首先要求 LLM 从实体-关系知识库中检索与当前测试模块相关的所有 SWS-ID 标识的规范条目。随后,引导模型分析并找到有关联性的规范。若多条规范存在逻辑依赖(如 SWS_Os_00002 与 SWS_Os_00407 共同约束调度表行为),则提取其复合测试目的;若为独立规范,则直接生成对应的单一测试目的。在此基础上,进一步指导 LLM 根据测试目的描述,从知识库中匹配显式配置项以及对应的隐式配置项,例如,为验证“调度表顺序执行”,需调度表容器、到期点子参数及相关的计数器容器等。

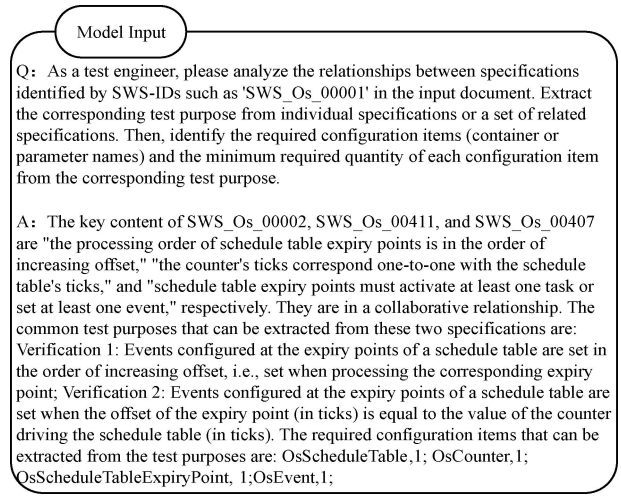

图 4　少样本思维链提示词

Fig. 4　Few-shot chain-of-thought prompting

值得注意的是,针对上述过程可能遗漏的隐式依赖配置项(如父容器的必填子项等),可通过后续步骤进一步补全。例如,当显式配置 OsScheduleTable 容器时,需根据配置规范自动补全其所需要的相关模块下的必要容器和参数。这种分层递进的生成策略,既保证了关键配置项的精准提取,又通过结构化知识库与生成算法的协同作用,显著降低了人工干预需求。最终,实现在复杂规范场景下测试数据生成效率与完整性的双重提升。

2.3　构建带属性的配置项树

对于类似 AUTOSAR 一致性测试中复杂的配置数据而言,其组成不仅依赖于规范中明确声明的显式配置项,还需解决隐式配置项的补全问题。这些隐式配置项包括未直接声明但由依赖关系衍生的必填子项以及跨模块引用等。例如,若仅按照规范显式要求配置了 OsScheduleTable 容器,而忽略了其必需的参数(如 OsScheduleTableDuration 等),将导致生成的 ARXML 配置数据文件无法通过

AUTOSAR 合规性验证。

为此,本研究提出了一种带属性的配置项生成树模型,通过结构化表示配置项的层级关系、自身属性及其与其他配置项之间的约束,实现了显式与隐式配置项的全覆盖。该生成树的构建基于 AUTOSAR 配置规范,整合了所有配置项的 name、Type、Multiplicity、Range 等属性。每个节点对应一个配置项,其中 Module 作为根节点,代表 AUTOSAR 模块(如 OS 模块);Container 作为中间节点,用于承载子容器或参数;Parameter 作为叶子节点,存放具体参数;Reference 类型节点则用于描述同一模块内不同节点或跨模块节点的关联关系,因此额外包含 Target 属性,用于指示引用的目标节点。例如,在 OS 模块下的 OsScheduleTable 容器中,参数 OsScheduleTableCounterRef 必须通过 Target 属性引用某个 OsCounter 容器,以定义调度表的时间基准。

Multiplicity 属性由下限重数(lowerMultiplicity)和上限重数(upperMultiplicity)共同定义,约束配置项允许的实例数量范围。例如,OsAppMode 容器的 lowerMultiplicity 为 1,表示在配置文件中必须至少定义 1 个应用模式;而 OsTask 容器的 upperMultiplicity 为 *,表示可根据系统需求灵活添加多个任务实例。对于参数类配置项,Range 属性用于限定参数的取值范围。例如,布尔参数的 Range 仅允许 True 或 False;整型参数(如 OsScheduleTableDuration)需满足[0,18 446 744 073 709 551 615]的取值区间;枚举型参数(如 OsTaskPriority)必须从预定义集合{LOW, MEDIUM, HIGH}中选择。该约束机制不仅确保了配置数据的合规性,还能通过自动化校验机制降低人工修正成本。

配置项树的构造流程如算法 1 所示。算法的入口函数 BuildAttributeTree 接收模块定义 *root_module* 作为输入。首先,通过 createTreeNode(*root_module*.name)创建根节点 *root*,并设置其 Module 类型及多重性约束。随后,算法在循环中遍历模块内的容器集合 C,每个容器配置项记为 c_i。对于每个 c_i,通过 createTreeNode(c_i.name)构建容器节点,并设置 Type 和 Multiplicity 属性;若容器为 Reference 类型,则进一步调用 *container_node*.set_attribute(Target)明确其引用目标。

此后,调用 ProcessContainerContent(c_i, *container_node*)进入容器内容处理过程。在函数 ProcessContainerContent(c_i, *parent_node*)中,首先执行循环遍历容器 c_i 的参数集合 P,每个参数记为 p_i。对于每个 p_i,算法通过 createTreeNode(p_i.name)创建参数节点 *param_node*,并注入 Type 和 Multiplicity 属性;若参数类型为{Integer, Boolean, Enum, Float}中的一个,则进一步为其设置合法取值范围。

接下来,算法在循环中递归处理容器 c_i 的子容器集合 SC,每个子容器记为 sc_i,通过 createTreeNode(sc_i.name)创建子容器节点,并调用 ProcessContainerContent 完成递归构建。最终,算法返回一棵完整的带属性的配置项树,其节点不仅记录了结构层级,还携带关键属性。

算法 1 构建带属性的配置项树

输入:模块配置项定义 *root_module*

输出:属性配置项树 *root*

```
Function BuildAttributeTree(root_module)
    root←createTreeNode(root_module.name)
    root.set_attribute(Type, Multiplicity)
    for c_i ∈ C do
        container_node ← createTreeNode(c_i.name)
        container_node.set_attribute(Type, Multiplicity)
        if c_i.type == 'Reference'
            container_node.set_attribute(Target)
        end if
        ProcessContainerContent(c_i, container_node)
    end for
end Function
Fuction ProcessContainerContent(c_i, parent_node)
处理参数节点
    for p_i ∈ c_i.P do
        param_node ← createTreeNode(p_i.name)
        param_node.set_attribute(Type, Multiplicity)
        if p_i.type in ['Integer','Boolean','Enum', 'Float']
            param_node.set_attribute(Range)
        end if
    end for
递归处理子容器
    for sc_i in c_i.SC do
        sub_node ← createTreeNode(sc_i.name)
        ProcessContainerContent(sc_i, sub_node)
    end for
end Function
```

通过这一结构化建模方法,配置树能够全面覆盖 AUTOSAR 规范中的显式与隐式配置项,为后续的配置生成、验证及一致性测试奠定基础。

2.4　配置数据 ARXML 文件生成

为生成符合 AUTOSAR 标准的完整配置数据,本节将详细阐述如何基于带属性的配置项生成树(由模块、容器、参数构成的层级结构),并通过动态结合显式需求与隐式依赖,实现自动化配置数据 ARXML 文件的生成。具体流程如图 5 所示,主要涵盖以下 6 个步骤。

1)输入解析与节点定位:用户将第 1 阶段从自然语言规范文档中生成的显示配置项(采用简化的“配置项名称,数量”格式)作为输入(例如“OsScheduleTable, 1; OsCounter,1”),通过_parse_explicit_input()方法解析为

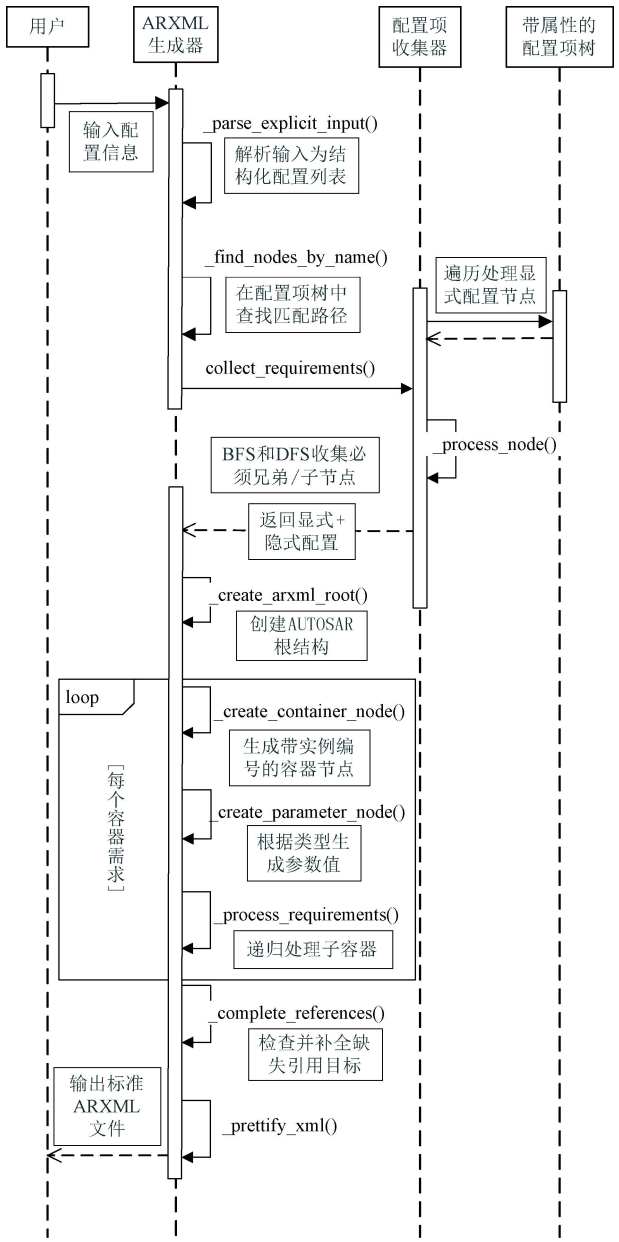

图 5　生成完整配置数据 ARXML 文件流程

Fig. 5　Flowchart for generating complete configuration data ARXML file

结构化的配置列表。随后，调用_find_nodes_by_name()方法，在带属性的配置项树中以广度优先搜索策略定位目标节点路径（如/Os/OsScheduleTable），确保准确地将用户意图映射至底层数据结构。

2）需求收集与隐式依赖分析：通过显式配置处理，利用 EnhancedConfigCollector 的 collect_requirements()方法遍历每个显式配置项。通过_process_node()方法启动双向遍历，既采用广度优先遍历同级节点，检查父节点下的所有兄弟容器。若其 Multiplicity. lower 值≥1（例如 OsAppMode 容器必须存在），则将其标记为隐式依赖。同时，采用深度优先遍历子节点，即递归扫描子容器及参数，对 Multiplicity. lower 值≥1 的必填参数和子容器进行隐式需求标记。

3）引用关系追踪：针对类型为 Reference 的配置项，记录其目标容器类型至 reference_map，以便后续处理。同时，统计引用源的 Multiplicity. lower 总和及来源容器数量，动态计算目标容器的最小实例需求。

4）ARXML 结构生成：首先进行根框架构建，即通过调用_create_arxml_root()函数创建符合 AUTOSAR 标准的 XML 根结构，其中包含命名空间定义及模块节点。接着进行容器实例化，利用_create_container_node()函数为每个需求容器生成带序号的实例节点（例如 OsScheduleTable_1），其＜SHORT-NAME＞标签按照“配置项名_序号”的规则进行命名，＜DEFINITION-REF＞则指向预处理树中的定义路径。最后进行参数值注入，根据参数类型调用_create_parameter_node()函数动态生成合规值——对于数值型参数如 Integer，会在指定 Range 范围内随机生成一个整数作为该配置项的值；而对于 Reference 类型的配置项，其值则为指向目标容器的完整路径。

5）引用完整性补全：通过调用_complete_references()方法，系统将扫描所有＜ECUC-REFERENCE-VALUE＞标签。若发现目标容器未包含在已生成的列表中，则会触发_generate_missing_container()方法，递归生成该容器及其必需的子项，以确保引用链的完整性。在此过程中，系统将自动继承目标容器的 Multiplicity 约束，并生成其必填参数和子容器。

6）标准化输出：最终通过调用_prettify_xml()方法，借助 xml. dom. minidom 对生成的 XML 树进行缩进与换行处理，生成符合 AUTOSAR XSD Schema 要求的可读文件。该文件的结构严格遵循“模块→容器→参数/子容器”的层级嵌套规则，且所有实例均通过＜SHORT-NAME＞和＜DEFINITION-REF＞实现与带属性配置项树定义的双向关联。

3　实验与分析

3.1　实验环境

本研究在数据生成阶段所采用的软硬件环境如表 2 所示。除了上述配置外，用于本实验的数据包括 15 个 AUTOSAR 基础软件模块（例如 OS、COM 即通信模块等）的规范文档，其版本为 AUTOSAR_SWS 4. 2. 2。

3.2　评估指标

本研究通过以下 5 个维度的量化指标，验证本文提出的配置数据自动化生成方法的有效性。

1）隐式依赖提取完整性

通过统计基于混淆矩阵的 4 个基本指标——真正例（true positive，TP）、假正例（false positive，FP）、真反例（true negative，TN）和假反例（false negative，FN），进而计

表 2　实验软硬件配置环境

Table 2　Experimental software and hardware configuration environment

环境	参数
GPU	RTX 4090
显存/GB	48
CPU	Intel(R) Xeon(R) Gold 6430 CPU
内存/GB	100
数据盘/GB	100
深度学习框架	Pytorch 2.3.0
编程语言	Python3.12
CUDA	12.1
操作系统	Ubuntu 22.04
LLM	Qwen2.5 32b

算精确率(precision,P)、召回率(recall,R)和 F_1 分数(F_1 score)3 项关键指标。

在本文中,TP 代表模型正确预测为正例的样本数量(即准确提取的隐式依赖项);FP 代表模型错误预测为正例的样本数量(即错误提取的隐式依赖项);FN 代表模型错误预测为反例的样本数量(即未提取到的隐式依赖项)。TP 与 FP 之和表示提取的所有隐式依赖项,而 TP 与 FN 之和则代表实际存在的所有隐式依赖项。P 指标则用于衡量所提取的隐式依赖项中,正确提取的比例,如式(1)所示。

$$P = \frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FP}} \times 100\% \tag{1}$$

R 指标反映了在所有隐式依赖项中,有多大比例被准确提取,如式(2)所示。

$$R = \frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FN}} \times 100\% \tag{2}$$

F_1 score 作为精确率和召回率的调和平均数,旨在全面评估方法的性能表现,如式(3)所示。

$$F_1\ \mathrm{score} = 2 \times \frac{PR}{P+R} \times 100\% \tag{3}$$

2)人工修正率(human fix rate,HFR)

在本文中,HFR 被定义为在所有生成的配置数据文件中,需人工修正的配置项数量占全部生成的配置项总数的百分比,如式(4)所示。

$$\mathrm{HFR} = \frac{\sum_{i=1}^{N} \mathrm{FC}_i}{\sum_{i=1}^{N} \mathrm{TC}_i} \times 100\% \tag{4}$$

其中,FC_i 代表在第 i 个文件中需要手动修正的配置项数量;TC_i 代表在第 i 个文件中生成的配置项总数;N 代表配置数据文件的总数。

3)覆盖率

本文将基于配置项覆盖率(configuration coverage rate,CCR)和规范覆盖率(specification coverage rate,SCR)这两项指标,对覆盖率的实际效果进行评估。

CCR 定义为生成的配置数据文件中覆盖的配置项数量与所有模块中配置项总数的比值,如式(5)所示。

$$\mathrm{CCR} = \frac{C_{\mathrm{cover}}}{C_{\mathrm{total}}} \times 100\% \tag{5}$$

其中,C_{cover} 代表生成的配置数据中覆盖的配置项数量;C_{total} 表示所有模块中定义的配置项总数。

SCR 定义为生成的配置数据所覆盖的规范数量与文档中总规范数量的比值,如式(6)所示。

$$\mathrm{SCR} = \frac{S_{\mathrm{cover}}}{S_{\mathrm{total}}} \times 100\% \tag{6}$$

其中,S_{cover} 代表用于生成配置数据的规范数量;S_{total} 代表输入文档中规范的总数。

4)时间效率提升率(time efficiency improvement rate,TEIR)

TEIR 用于衡量自动化配置数据生成方法相较于人工方法在生成单个配置数据文件时的效率提升程度,如式(7)所示。

$$\mathrm{TEIR} = \frac{T_{\mathrm{manual}} - T_{\mathrm{auto}}}{T_{\mathrm{manual}}} \times 100\% \tag{7}$$

其中,T_{manual} 代表人工生成单个配置数据文件的平均时间;T_{auto} 代表自动化方法生成单个配置数据文件的平均时间[22]。

5)测试场景生成增长率(test scenario generation rate,TSGR)

TSGR 用于评估在相同规范条件下,自动化方法相较于人工方法生成的测试场景数量的增长比例,如式(8)所示。

$$\mathrm{TSGR} = \frac{N_{\mathrm{auto}} - N_{\mathrm{manual}}}{N_{\mathrm{manual}}} \times 100\% \tag{8}$$

其中,N_{manual} 代表人工方法基于规范文档生成的测试场景总数;N_{auto} 则代表自动化方法基于相同规范生成的测试场景总数。

3.3　实验结果与分析

本研究将基于对比实验和消融实验的结果展开分析。对比实验旨在评估在相同输入规范文档的前提下,本文提出的配置数据自动化生成方法在各项评估指标上与其他 3 种自动化方法的对比效果。消融实验则通过对本文方法进行模块拆解,评估各模块对整体性能的具体影响。

1)对比实验

本文选取文献[7-8,11]中提出的方法作为对比对象,主要基于其在自动化测试数据生成领域中所代表的不同主流技术路线及其与本文研究问题的相关性。文献[8]基于知识图谱技术对规范中的实体及关系进行结构化表示,通过规则推导测试数据;文献[7]通过结构化语义抽取与约束建模将自然语言规范转换为形式化表示;文献[11]则

利用大语言模型从自然语言规范直接生成结构化配置描述，代表了数据驱动方法在该领域的研究进展。

与上述方法相比，本文方法同样以自然语言规范为输入，但通过引入图检索增强生成机制、基于 Few-Shot CoT 的推理策略以及带属性的配置项树结构，实现了对跨规范上下文关系与隐式配置的自动推导，从而在配置数据生成的质量与自动化程度上进行了针对性改进。因此，选择上述方法作为对比对象，旨在从不同技术路线对本文方法进行综合评估。将文献[7-8,11]和本文方法依次分别标记为 M_1、M_2、M_3 和 M_4。实验结果如表 3 所示。

表 3　不同方法对比实验结果

Table 3　Experimental comparison results of different methods　%

方法	P	R	F_1 score	HFR	CCR	SCR	TEIR	TSGR
M_1	97.20	85.80	91.14	16.20	93.50	100.00	51.20	1.50
M_2	96.50	80.40	87.72	23.00	91.00	100.00	36.01	2.00
M_3	96.15	43.70	60.09	45.10	62.00	80.05	**80.12**	0.20
M_4	**98.00**	**97.03**	**97.51**	**5.00**	**99.50**	**100.00**	71.40	**3.00**

从表 3 中可以看出，尽管本文方法 M_4 与 M_1 方法都采用了先将自然语言描述的规范文档转化为结构化领域知识库的策略，但由于 M_1 方法采用的是人工构建的方式，导致其在 TEIR 指标上的表现不佳。

此外，M_1 方法基于领域知识库，并结合历史测试数据与语义相似度，推荐与规范相匹配的测试子路径，随后直接进行下一步测试数据的填充。在此过程中，领域知识库的完整性和历史测试数据的丰富性均会影响隐式依赖项（即规范中未显式声明的隐式数据）提取的完整性。M_2 方法在 TEIR 指标上表现较低的原因在于它不仅需要人工将规范预处理成 RUCM 形式，还需手动创建领域模型，且在实体缺失时需工程师补充。其次，面对复杂且模糊的规范描述，M_2 方法通过 NLP 生成的 OCL 约束往往不准确，甚至无效，因为无法有效提取该场景下的隐式依赖，从而导致隐式依赖相关指标及 CCR 降低。M_3 方法除 TEIR 指标较高外，其余评估指标均较低，主要原因是其训练集单一，目前仅对 AUTOSAR 中的 CanNM 模块有效，缺乏可扩展性。

在 TSGR 指标上，由于 M_2 方法采用了 3 种覆盖策略（分支覆盖、定义-使用覆盖、子类型覆盖）来多角度共同生成测试路径（每条测试路径即为 1 个测试场景），因此 M_2 方法的 TSGR 较高。而本文 M_4 方法测试场景增长率最高的原因在于对提取到的实体、关系及相关节点进行了聚类分析，从而能够挖掘潜在的规范关联，进一步生成多样化测试目的下对应的测试场景。例如，在利用本文方法对 OS 模块进行一致性测试的测试目的生成过程中，发现了以下典型但人工难以发现的测试场景，并为后续的源代码错误定位奠定了基础。场景 1：由规范[SWS_Os_00033]、[SWS_Os_00553]和 OSEK/VDX Operating System 文档中关于资源的描述推导的测试目的是“验证：1. 当类别 2 中断占用资源并超出预设的资源锁定预算时，操作系统是否设置 E_OS_PROTECTION_LOCKED 为错误代码来调用 ProtectionHook()；2. 当 ProtectionHook()返回 PRO_TERMINATETASKISR 时，是否正确终止了该类别 2 中断服务程序，同时确保系统的其他功能如任务能正常运行。”通过在一致性测试中验证该测试场景得知，验证中的第 1 点没有发现问题，而第 2 点中的“确保系统的其他功能如任务能正常运行”不能正常实现。经过相关人员定位分析后发现，原因在于虽然成功终止了中断程序，但并没有解除中断屏蔽寄存器，从而导致在 Cortex-M3/M4 处理器下 PendSV 这个用于任务切换的低优先级中断无法被响应，导致任务调度延迟。场景 2：由规范[SWS_Os_00064]推导的测试目的是“验证：当任务（自启动任务、调度表激活的任务、警报激活的任务、基本任务、扩展任务）的 OsTaskExecutionBudget 被耗尽时，操作系统模块调用 ProtectionHook()并传入 E_OS_PROTECTION_TIME。”通过该场景的一致性验证后发现，在以上多种类型的任务中，首个自启动任务在执行超时后，时间保护超时处理并未开启。场景 3：由规范[SWS_Os_00466]和 OSEK/VDX Operating System 中关于警报的描述推导的测试目的是“验证在警报中，在 OsTaskTimeFrame 结束前激活任务，则操作系统模块不应执行激活操作，并应调用 ProtectionHook()，传入 E_OS_PROTECTION_ARRIVAL 参数。”经上述场景的验证后发现，原始代码允许上一次被激活后但未结束的任务被警报再一次激活，即它忽略了对任务频繁调用的保护。

2）消融实验

为验证本文中各项改进策略对自动化配置数据生成性能的独立有效性，因此通过消融实验对本文方法的关键模块进行了单一效果对比验证。具体而言，实验以本文提出的完整方法 M_4 作为基线模型，该模型包含本文方法的全部功能模块，可作为衡量各模块贡献的标准。消融实验通过逐一移除本文方法中的单个模块，并保持其余组件不变，观察模型在各项评价指标上的性能变化，从而分析对应改进策略对整体性能的影响。

为评估本文方法中各核心模块对整体性能的独立

贡献,根据方法结构设计,构造 3 组消融模型:分别移除 M_4 方法中的 Few-Shot CoT 提示词模块、领域知识库构建模块以及带属性的配置项树模块,对应的实验组分别记为 M_4-P、M_4-K、M_4-T。各消融模型的具体修改方式及其关键影响分析方向如表 4 所示,实验结果如表 5 所示。

表 4　消融实验组

Table 4　Ablation experiment groups

实验组	修改描述	关键影响分析方向
无 Few-Shot CoT(M_4-P)	禁用少样本思维链提示词,采用零样本直接生成测试目的与配置项	复杂任务逻辑推理与领域任务解答能力
无构建知识库(M_4-K)	跳过知识库构建步骤,直接基于原始规范文档文本进行测试目的与配置项生成	多跳推理与长上下文中跨规范关联分析能力
无属性配置项树(M_4-T)	移除构造属性配置项生成树,仅通过 LLM 提取的配置项直接生成配置数据	配置项层级结构与隐式依赖完整性保障

表 5　不同实验组与基线组的实验结果对比

Table 5　Experimental results comparison between different groups and the baseline group　%

对比组别	P	R	F_1 score	HFR	CCR	SCR	TEIR	TSGR
M_4-P	95.20	76.10	84.59	29.00	94.01	100.00	64.50	1.00
M_4-K	94.10	65.05	76.92	38.50	88.16	100.00	75.16	0.40
M_4-T	96.80	82.00	88.79	18.02	95.00	83.00	60.00	3.00
M_4	**98.00**	**97.03**	**97.51**	**5.00**	**99.50**	**100.00**	**71.40**	**3.00**

从表 5 中可以看出,完整模型 M_4 方法在所有指标上均取得最佳表现,显著优于任何单独移除模块后的模型版本。而另外 3 组消融实验在不同程度上对各项评估指标产生了负面影响。

M_4-P 方法通过零样本来应对本文所述的领域任务(即从单个或多个组合规范中提取测试目的及所需配置项)。然而,由于模型缺乏显式推理链的构造示例,这不仅显著削弱了其在领域任务上的准确理解能力和推理能力,导致其容易直接给出不完整或错误的配置项,还降低了该方法生成的测试场景数量。尽管如此,得益于 M_4-P 方法构建了领域知识库,其 TSGR 指标仍高于 M_4-K 方法。M_4-K 方法直接利用 RAG 框架整合规范文档后进行领域任务的检索和生成,导致其无法从规范的实体-关系结构中进行跨段落、多跳检索,无法利用规范间的上下文关联信息,无法准确且完整地生成所需的显式配置项,使得 LLM 在任务回答的深度和广度上都受到了影响,这也是它相较于 M_4-P 和 M_4-T 方法在几乎各项指标上表现更差的原因。M_4-T 方法不再使用构造的带属性的配置项树来补充和完善生成的配置项,也不再进行配置数据的结构补全,而是让 LLM 直接输出配置项,再简单映射为 ARXML 结构,这在一定程度上降低了配置数据的生成质量,而且由于其缺乏隐式配置项的补全,导致配置项规范的覆盖率降低,SCR 指标下降最为显著。消融实验的结果系统地证明了本文所提出的 M_4 方法中的 3 个核心模块对自动化配置数据生成流程的必要性与显著贡献。

3.4　案例分析

本节以 AUTOSAR OS 模块完整规范文档作为输入,演示本文方法在真实规范场景下从“规范解析与关联发现”到“配置数据补全并生成 ARXML”的完整流程。系统首先对规范文档进行结构化解析并构建领域实体—关系知识库,在此基础上对“调度表(schedule table)”相关规范进行关联检索,选取一个具有代表性的场景:单次(single shot)调度表在处理完最终到期点(final expiry point)后,经过最终延迟(final delay)滴答停止处理。针对该场景,系统生成的测试目的为:验证在最终到期点经过最终延迟滴答数后,单次调度表的状态是否为 SCHEDULETABLE_STOPPED。

该测试目的的形成需要闭合“行为约束—时间计量—状态判定”3 类信息:1)规范 [SWS_Os_00009] 给出单次调度表在最终到期点后经过最终延迟停止处理的行为;2)规范 [SWS_Os_00411] 给出计数器时钟滴答(counter tick)与调度表时钟滴答(schedule table tick)的一致性,使“等待最终延迟”具有可操作的计量基准;3)调度表状态定义(调度表不处于活动状态,即不被操作系统处理时,其状态为 SCHEDULETABLE_STOPPED)将“停止处理”映射为可检验的状态断言。为保证该测试目的可落地为可执行实例,系统同时引入调度表结构与规范的约束用于约束实例化过程,例如:调度表至少包含一个到期点(expiry

point)、到期点具有偏移量(offset)、偏移量唯一且按递增顺序处理、每个到期点至少包含一次任务激活或事件设置,以及最终延迟和相邻到期点延迟受计数器取值范围约束等。

本文方法阶段 1 的输出如清单 1 所示,作为带属性配置项树的直接输入。清单 1 包含:module 表示目标 AUTOSAR 模块;test_objective 为生成的测试目的;sws_refs 记录支撑该测试目的与配置项的相关规范;explicit_items 给出需实例化的显式配置项及其数量;params 汇总隐式容器/参数的可执行取值与数量策略以及规范依据,其中 v 为具体取值(供带属性配置项树直接使用)、ref 为依据来源、strategy 为取值/实例化的简短说明用于人工回溯。第 2 阶段读取清单 1 后,依据带属性的配置项树模型自动完善必需的隐式容器/参数/子容器,依据引用关系闭合引用链(如 CounterRef→OsCounter、ActivateTaskRef→OsTask),并在满足约束的前提下生成对应的配置项数值,最终输出 ARXML 配置数据文件,如清单 2 所示。

清单 1 测试目的与配置需求结构化输出

输入:OS 规范文档

输出:测试目的、配置项、实例化要点 JSON

```
{
  "module": "Os",
  "test_objective":"Verify that the status of single shot
ScheduleTable is SCHEDULETABLE_STOPPED,
after Final Delay Ticks have elapsed from the Final
Expiry Point.",
  "sws_refs": ["SWS_Os_00009", "SWS_Os_00411",
"ScheduleTable state definition: not active =>
SCHEDULETABLE_STOPPED"],
    "explicit_items":{"OsScheduleTable":1,
"OsCounter": 1, "OsTask": 2,
"OsScheduleTableExpiryPoint": 2},
    "params": {
      "OsScheduleTableRepeating": {
        "v": false,
        "ref": ["SWS_Os_00009",
      "ECUC:OsScheduleTableRepeating"],
        "strategy":"Interpret 'single-shot' as non-
      repeating; set OsScheduleTableRepeating=false."
    },
      …
  }
```

清单 2 ARXML 配置数据文件输出

输入:测试目的、配置项、实例化要点 JSON

输出:ARXML 配置数据文件

```
<ECUC-CONTAINER-VALUE>
  <SHORT-NAME>OsCounter_1</SHORTNAME>
  …
  <VALUE>45</VALUE>
  <! --OsCounterMaxAllowedValue -->
  <VALUE>3</VALUE>
  <! -- OsCounterMinCycle -->
  <VALUE>1</VALUE>
  <! -- OsCounterTicksPerBase -->
   <VALUE>SOFTWARE</VALUE>
   <! -- OsCounterType -->
</ECUC-CONTAINER-VALUE>
<ECUC-CONTAINER-VALUE>
  <SHORT-NAME>OsScheduleTable_1
  </SHORT-NAME>
  …
  <VALUE>25</VALUE>
  <! -- OsScheduleTableDuration -->
  <VALUE>0</VALUE>
  <! -- OsScheduleTableRepeating=false -->
  <VALUE-REF>/Ecuc/Os/OsCounter_1
  </VALUE-REF> <! -- CounterRef -->
  <SUB-CONTAINERS>
    …
  </SUB-CONTAINERS>
</ECUC-CONTAINER-VALUE>
…
```

4　结　　论

本研究针对 AUTOSAR 一致性测试的配置数据生成过程中自然语言规范解析困难、隐式依赖提取不完整等问题,提出了一种基于图检索增强生成框架的自动化生成方法。该方法通过整合检索增强生成、领域知识库、少样本思维链提示词及带属性的配置项生成树,实现了从复杂模糊规范到合规配置数据 ARXML 文件的高效转化。实验结果表明,相较于现有方法,本文提出的方法在隐式依赖提取完整性、人工修正率及覆盖率方面均取得了显著进步,为车载嵌入式软件的自动化测试提供了可靠的解决方案。

本研究的核心贡献主要体现在以下 3 个方面:首先,创新性地将图检索增强生成技术引入 AUTOSAR 领域,通过动态知识检索有效缓解了 LLM 的幻觉问题;其次,设计了 Few-Shot CoT 提示词,引导 LLM 从知识库中精准提

取测试目的与所需配置项;最后,提出了基于 BFS 和 DFS 的遍历策略,有效补全和完善隐式配置项依赖。这些技术不仅适用于 AUTOSAR,还可扩展至其他复杂标准中,具有较高的工程普适性。

然而,本研究仍存在一定的局限性,即在完全自动化生成完整测试用例方面尚未进行深入研究。后续工作将进一步探索上述问题,以全面提升一致性测试中的智能化水平。

参考文献

[1] 陈博言, 沈晴霓, 张晓磊, 等. 智能网联汽车的车载网络攻防技术研究进展[J]. 软件学报, 2025, 36(1): 341-370.
CHEN B Y, SHEN Q N, ZHANG X L, et al. Research progress on attacks and defenses technologies for in-vehicle network of intelligent connected vehicle[J]. Journal of Software, 2025, 36(1): 341-370.

[2] FÜRST S, BECHTER M. AUTOSAR for connected and autonomous vehicles: The AUTOSAR adaptive platform[C]//2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshop(DSN-W), 2016: 215-217.

[3] 陈灿, 杨兴达, 方菱. 基于决策表的 AUTOSAR 操作系统一致性测试研究[J]. 计算机工程与科学, 2023, 45(4): 622-629.
CHEN C, YANG X D, FANG L. AUTOSAR operating system conformance test research based on decision table[J]. Computer Engineering & Science, 2023, 45(4): 622-629.

[4] 张建, 王辉, 李超超, 等. 基于时间约束的 CAN 网络管理一致性测试[J]. 电子测量技术, 2024, 47(7): 19-27.
ZHANG J, WANG H, LI C C, et al. Conformance testing of CAN network management based on time constraints[J]. Electronic Measurement Technology, 2024, 47(7): 19-27.

[5] 蔡一涵, 马立鹏, 杨卫东, 等. 针对需求缺陷检测任务的自然语言需求数据集评估[J]. 计算机应用与软件, 2024, 41(11): 78-85.
CAI Y H, MA L P, YANG W D, et al. Natural language requirement dataset evaluation for requirement defect detection task [J]. Computer Applications and Software, 2024, 41(11): 78-85.

[6] YADAV A, PATEL A, SHAH M. A comprehensive review on resolving ambiguities in natural language processing[J]. AI Open, 2021, 2: 85-92.

[7] 郑台台, 姚燕, 蔡晋辉. 基于三维坐姿的压力数据自动标注方法[J]. 仪器仪表学报, 2023, 44(10): 71-79.
ZHENG T T, YAO Y, CAI J H. Automatic annotation method for pressure data based on three-dimensional sitting posture[J]. Chinese Journal of Scientific Instrument, 2023, 44(10): 71-79.

[8] WANG C H, PASTORE F, GOKNIL A, et al. Automatic generation of acceptance test cases from use case specifications: An NLP-based approach[J]. IEEE Transactions on Software Engineering, 2022, 48(2): 585-616.

[9] 臧远山. 基于知识图谱的 CTCS-3 级列控系统车载设备测试数据自动生成[D]. 北京:北京交通大学, 2023.
ZANG Y S. Automatic generation of on-board equipment test data for CTCS-3 train control system based on knowledge graph [D]. Beijing: Beijing Jiaotong University, 2023.

[10] WANG R, LIU S Y, SATO Y J. SIT-SE: A specification-based incremental testing method with symbolic execution [J]. IEEE Transactions on Reliability, 2021, 70(3): 1053-1070.

[11] SHIGIHARA K, HONDA S, TAKADA H. Test program generator for AUTOSAR OS[C]//2017 13th European Dependable Computing Conference(EDCC), 2017: 79-86.

[12] EI-GNAINY N, SHANOUDA M, ESSAM A, et al. AI-enhanced AUTOSAR configuration: Efficient methods for dataset generation and automated code production[C]//2024 IEEE 8th Forum on Research and Technologies for Society and Industry Innovation(RTSI), 2024: 214-219.

[13] AUTOSAR GBR. Specification of operating system R4.2.2[S]. AUTOSAR GBR, 2015.

[14] EDGE D, TRINH H, CHENG N M, et al. From local to global: A graph RAG approach to query-focused summarization [J]. ArXiv preprint arXiv: 2404.16130, 2024.

[15] 彭宇, 季拓, 郭楚亮. 故障预测与健康管理领域大语言模型:应用与展望[J]. 仪器仪表学报, 2025, 46(10): 2-21.
PENG Y, JI T, GUO C L. Prognostics and health management domain-specific large language models: Applications and prospects[J]. Chinese Journal of Scientific Instrument, 2025, 46(10): 2-21.

[16] 徐智聪. AUTOSAR 中 BSW 运行时环境代码生成工具的设计与实现[D]. 北京:北京邮电大学, 2024.
XU Z C. Design and implementation of BSW runtime environment code generation tool in AUTOSAR[D]. Beijing: Beijing University of Posts and Telecommunications, 2024.

[17] 申振强. AUTOSAR 中 SWC 运行时环境配置及生成工具的设计与实现[D]. 北京:北京邮电大学, 2024.
SHEN Z Q. Design and implementation of the SWC runtime environment configuration and generating tools in AUTOSAR[D]. Beijing: Beijing University of Posts and Telecommunications, 2024.

[18] WANG B, XU C, ZHAO X M, et al. MinerU: An open-source solution for precise document content extraction[J]. ArXiv preprint arXiv:2409.18839, 2024.

[19] LEWIS P, PEREZ E, PIKTUS A, et al. Retrieval-augmented generation for knowledge-intensive NLP tasks[J]. Advances in Neural Information Processing Systems, 2020, 33: 9459-9474.

[20] VINYALS O, BLUNDELL C, LILLICRAP T, et al. Matching networks for one shot learning[J]. ArXiv preprint arXiv:1606.04080, 2016.

[21] WEI J, WANG X Z, SCHUURMANS D, et al. Chain-of-thought prompting elicits reasoning in large language models[J]. Advances in Neural Information Processing Systems, 2022, 35: 24824-24837.

[22] 刘行谋, 魏钞, 杨辉, 等. 汽车电控齿轮泵车载工况故障诊断方法研究[J]. 电子测量与仪器学报, 2025, 39(5): 227-240.
LIU X M, WEI C, YANG H, et al. Research on the fault diagnosis method of automotive electronically controlled gear pump under on-board working conditions[J]. Journal of Electronic Measurement and Instrumentation, 2025, 39(5): 227-240.

作者简介

王辉,硕士研究生,主要研究方向为车载软件测试技术。
E-mail:Q22201340@stu.ahu.edu.cn

方菱(通信作者),博士,副研究员,主要研究方向为嵌入式系统安全与测试。
E-mail:fangl@hfcas.ac