软件学报 ISSN 1000-9825, CODEN RUXUEW
2026,37(8):3052−3088 [doi: 10.13328/j.cnki.jos.007595] [CSTR: 32375.14.jos.007595]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

SmartGen-AADL: 多智能体系统需求分析与 AADL 模型生成*

葛楚妍, 王培远, 王甜甜, 黄钇茗, 杨小天

(哈尔滨工业大学 计算学部, 黑龙江 哈尔滨 150001)

通信作者: 王甜甜, E-mail: wangtiantian@hit.edu.cn

摘　要: 嵌入式系统建模是基于模型的软件开发的重要组成, 体系结构分析与设计语言 (architecture analysis and design language, AADL) 因其形式化表达软硬件结构与交互关系的能力, 广泛用于架构设计. 大语言模型 (large language model, LLM) 为从自然语言需求生成架构模型提供了新路径. 然而, 现有模型在需求语义理解、AADL 组件边界识别与连接关系建构等方面仍存在显著不足, 限制了其实用性与生成质量. 为解决上述问题, 提出一种面向嵌入式系统的智能建模方法——SmartGen-AADL, 整体方法基于多智能体协同机制构建, 融合语义解析、结构识别与提示增强生成等关键技术, 实现从自然语言需求到结构化 AADL 模型的高质量转换. 方法核心包括 3 个阶段: 首先, 系统通过结构化智能体完成系统架构文档中系统架构的识别与标准化需求语句的提取; 随后, 子问题智能体基于条目级分析与组件交互挖掘, 实现对需求粒度的细化与交互关系的显式建模; 最后, 构件生成智能体在语义提示中融合结构引导与基于检索增强生成 (retrieval-augmented generation, RAG) 的相似组件检索结果, 引导 LLM 生成符合 AADL 语法规范的组件代码. 为支撑上述流程, 构建了“条目化需求-AADL 组件”知识库以及“系统架构文档-AADL 架构”语义对齐数据集. 在 15 个嵌入式系统应用场景上的实验结果表明, 相较于仅依赖简单提示工程的方法, 所提多智能体协同建模方法在 4 个主流大语言模型上均展现出显著优势. 其中, 在 DeepSeek-r1 模型上的提升最为突出: 组件错误行占比平均降低 34.37%, F_{BERT} 语义相似度平均提升 6.21%, 结构匹配度提升超过 20%, 人工评分整体提高约 0.7 分. 进一步的消融实验结果显示, 子问题识别机制增强了对建模粒度的控制能力; 系统结构树构建提供了组件组织与层级拓扑信息; 检索增强生成机制为模型提供了外部知识支撑并降低了幻觉率; 通信连接识别确保了模型的接口完备性与交互闭环, 四者协同促进了自然语言到 AADL 建模语言对齐与模型一致性的显著提升.

关键词: 结构分析与设计语言 (AADL); 需求分析; 多智能体; 系统工程

中图法分类号: TP311

中文引用格式: 葛楚妍, 王培远, 王甜甜, 黄钇茗, 杨小天. SmartGen-AADL: 多智能体系统需求分析与AADL模型生成. 软件学报, 2026, 37(8): 3052–3088. http://www.jos.org.cn/1000-9825/7595.htm

英文引用格式: Ge CY, Wang PY, Wang TT, Huang YM, Yang XT. SmartGen-AADL: Multi-agent-driven System Requirements Analysis and AADL Model Generation. Ruan Jian Xue Bao/Journal of Software, 2026, 37(8): 3052–3088 (in Chinese). http://www.jos.org.cn/1000-9825/7595.htm

SmartGen-AADL: Multi-agent-driven System Requirements Analysis and AADL Model Generation

GE Chu-Yan, WANG Pei-Yuan, WANG Tian-Tian, HUANG Yi-Ming, YANG Xiao-Tian

(Faculty of Computing, Harbin Institute of Technology, Harbin 150001, China)

* 基金项目: 国家自然科学基金 (61977020); 工信部高质量发展专项 (CEIEC-2024-ZM02-0067); 黑龙江省自然科学基金 (LH2019F046); 黑龙江省重点研发计划 (JD2023SJ21); 哈尔滨市科技计划 (2022ZCZJCG019); 深圳市关键技术攻关项目 (JSGG2021110892802003)
本文由“大模型与软件工程”专题特约编辑聂长海教授、王璐副教授、王莹教授、姜艳杰副教授、张敏灵教授推荐.
收稿时间: 2025-09-08; 修改时间: 2025-10-28; 采用时间: 2025-12-08; jos 在线出版时间: 2025-12-24
CNKI 网络首发时间: 2026-03-19

Abstract: Modeling embedded systems is an essential component of model-based software development. The architecture analysis and design language (AADL), with its ability to formally express hardware-software structures and interaction relationships, is widely applied in system design. Large language models (LLMs) provide a new pathway for generating architecture models from natural language requirements. However, existing approaches exhibit significant limitations in requirement semantic understanding, boundary identification of AADL components, and construction of connection relationships, which constrain their practicality and the quality of generated models. To address these challenges, this study proposes an intelligent modeling approach for embedded systems, termed SmartGen-AADL. The overall framework is built upon a multi-agent collaboration mechanism, integrating key techniques such as semantic parsing, structural recognition, and prompt-enhanced generation, thus enabling high-quality transformation from natural language requirements into structured AADL models. The method consists of three core stages: (1) a structural agent identifies system architectures from system architecture documents and extracts standardized requirement statements; (2) a sub-problem agent performs item-level analysis and interaction mining to refine requirement granularity and explicitly model component interactions; (3) a component generation agent incorporates structural guidance and retrieval-augmented generation (RAG) of similar components into semantic prompts, guiding the LLM to produce component code that conforms to AADL syntax. To support this process, a knowledge base of "itemized requirements-AADL components" and a semantic alignment dataset of "system architecture documents-AADL architectures" are constructed. Experimental results on 15 embedded system application scenarios demonstrate that, compared with approaches solely relying on prompt engineering, the proposed multi-agent collaborative modeling method achieves significant improvements across four mainstream LLMs. Among them, the performance gains are most pronounced on the DeepSeek-r1 model: the component line error rate is reduced by an average of 34.37%, F_{BERT} semantic similarity is increased by 6.21%, structural matching accuracy improves by more than 20%, and human evaluation scores rise by approximately 0.7 points. Furthermore, results from the ablation study reveal that the sub-problem identification mechanism enhances control over modeling granularity. The system structure tree contributes to component organization and hierarchical topology information. The retrieval-augmented generation mechanism supplies external knowledge support and reduces hallucination. Communication connection recognition ensures interface completeness and closed interaction loops. The synergy of these four mechanisms substantially promotes alignment between natural language requirements and the AADL modeling language, thereby improving model consistency.

Key words: architecture analysis and design language (AADL); requirements analysis; multi-agent; systems engineering

基于模型的分析方法已成为复杂嵌入式系统设计与验证中的关键技术手段. 在众多建模语言中, AADL (architecture analysis and design language, 结构分析与设计语言) 因其能够形式化描述软硬件组件及其交互关系的能力而被广泛采用. 现有研究中针对 AADL 系统的建模验证[1–3]和扩展[4–6]做了很多研究. 在 AADL 模型的构建过程中, 开发者通常需要根据系统的功能逻辑对组件进行模块化划分, 应用自顶向下的方式[7], 并在此基础上定义组件之间的接口与交互行为[8]. 该过程高度依赖需求文档提供的系统功能和约束信息, 因此需求的结构化表达直接关系到模型构建的质量与效率. 由于该流程耗时且重复性强, 当前研究逐渐开始探索利用大语言模型 (large language model, LLM) 从自然语言需求中自动生成 AADL 模型的可行性. LLM 具备强大的语言理解与生成能力, 能够在一定程度上减轻人工解析和建模的负担, 提高建模效率与准确性. 然而, 为了确保自动生成的模型具备良好的结构性和语义一致性, 条目化的需求表示成为这一过程的关键前提.

然而, 在实践中, 无论是公共软件项目的需求文档, 还是来自工业界的私有需求文档, 往往存在一个普遍问题: 大量需求被以自然语言段落的形式描述, 其中混合了多个相关的需求项, 却缺乏明确的分隔或标注. 这种描述方式违背了 ISO/IEC/IEEE 29148:2018(E) 标准中提出的"单一性"原则, 即每条需求应表述单一的能力、特性或约束. 文献 [9] 中称这种现象为"需求粘连 (requirement adhesion)", 其在 AADL 建模过程中带来了显著挑战. 首先, 混合表达的需求缺乏明确边界, 使得开发者难以准确识别独立的功能单元, 从而影响系统功能的模块划分与组件职责的明确性, 降低了模型的可维护性与复用性. 其次, 多项需求共存于同一段落中, 掩盖了它们之间的语义层次与依赖关系, 阻碍了功能结构的梳理与组件交互的抽取. 这不仅增加了模型构建的人工成本, 也限制了自动化建模工具在需求驱动模型生成与验证中的应用能力, 最终影响系统架构设计的质量与效率. 此外, 目前在 AADL 自动建模领域仍面临重要的基础资源瓶颈: 缺乏高质量的对齐数据集. 一方面, 现有公开数据集多聚焦于自然语言需求本身, 尚未形成需求条目与 AADL 组件之间明确对应关系的数据资源, 使得模型难以学习从需求到组件定义的映射规则. 另一方面, 也缺乏系统架构设计文档与其对应的 AADL 系统模型之间的标注数据, 限制了 LLM 在复杂系统结构还原、组件层级分析及接口关系抽取等任务中的训练与评估能力. 因此, 构建涵盖从需求、架构文档到 AADL

模型的多层次对齐数据集, 将成为推动该领域研究深入发展的关键基础工作.

除此之外, 自动化建模还面临组件间连接关系分析不足的问题. 现有方法多聚焦于单条软件需求的提取, 缺乏对其功能属性, 交互意图及其与其他需求关系的深入分析, 难以支持精确的组件互联建模. 事实上, 需求往往并非孤立存在, 而是与同一系统或项目中的其他需求存在依赖、细化或冗余等多种关联[10], 这些关系直接影响项目的排期, 范围划定与设计决策. 忽视这些关联不仅可能导致需求冲突和冗余, 还会影响建模的完整性与后续开发效率. 以 NASA JPL 的功能分解文档为例, 某些关键系统功能往往以分散, 间接的方式出现于多个段落中, 彼此间通过“先后顺序”“共享数据”“条件触发”等方式形成隐含联系[11]. 若仅进行句子级提取与孤立匹配, 忽视上下文语义与内在逻辑结构, 极易造成组件建模中的功能割裂与交互缺失. 为有效应对上述挑战, 不仅需要从方法层面提升自动建模过程中的语义理解与结构还原能力, 也亟须在数据层面构建覆盖“需求语义-系统结构-建模单元”多层信息对齐的数据集, 作为支撑建模算法训练与验证的基础. 然而, 通过对近年来 AADL 相关研究文献以及 GitHub、Zenodo 等主流开源平台的调研发现, 目前尚缺乏能够将自然语言系统架构文档与 AADL 架构模型进行一一映射的公开数据资源. 这一数据缺口已成为制约基于 LLM 的架构建模方法深入发展的关键瓶颈.

为解决这些问题, 本文首先构建了“条目化需求-AADL 组件”的知识库以及“系统架构文档-AADL 架构”语义对齐的数据集. 在此基础上, 针对自然语言需求向 AADL 架构模型转换过程中存在的语义理解不清及交互关系缺失等问题, 提出了一种融合多智能体协同机制的自动建模方法——SmartGen-AADL. 该方法依托多层语义对齐数据集, 提升了模型生成结果的准确性与工程可用性. 方法包含 3 个核心模块: 系统架构文档结构化模块, 由智能体负责提取系统架构信息与生成标准化需求条目; 子问题分析模块通过语义细化识别子问题并挖掘模块间交互关系与接口定义, 实现多层级系统理解; AADL 组件生成模块结合大语言模型与知识库检索生成具备功能语义与结构约束的 AADL 组件及连接关系, 形成从语义理解到架构建模的闭环流程. 本文主要贡献如下.

(1) 提出一种基于多智能体协同的自动建模框架 SmartGen-AADL, 支持从自然语言需求到高质量 AADL 模型的自动生成. 设计了涵盖子问题提取、结构树构建与组件连接识别的多阶段生成机制, 提升了模型在任务粒度控制与结构拓扑建模方面的能力, 并结合检索增强生成策略, 显著提高了复杂嵌入式系统架构的语义一致性与工程可部署性.

(2) 构建系统架构文档-AADL 架构对齐的数据集, 支持结构化生成流程的设计与验证.

(3) 构建超过 500 条的 AADL 组件-条目化需求的知识库, 作为智能体检索增强生成的基础支撑.

(4) 在 15 个典型系统的 AADL 架构上开展实验, 选取 DeepSeek-r1、Claude-3.7、ChatGPT-4o 与 Llama-4 这 4 个主流开源大语言模型作为对比基线, 比较应用本文方法前后的性能差异. 同时, 对 DeepSeek-r1 和 Claude-3.7 进一步开展消融实验, 从代码正确性、结构匹配度、语义相似度及专家评分等维度综合评估方法的有效性.

本文第 1 节对需求结构化提取与 AADL 建模的相关研究现状进行综述, 为本文研究提供理论与方法背景. 第 2 节阐述研究动机, 明确本研究所聚焦的问题及其意义. 第 3 节介绍数据集的构建流程, 包括原始数据的整理、预处理与标准化步骤. 第 4 节详细描述所提出的系统架构文档分析与 AADL 建模方法. 第 5 节说明实验设置, 包括实验数据、评价指标及基准模型、实验流程. 第 6 节呈现实验结果并进行分析, 以验证所提方法的有效性与优势. 第 7 节讨论研究可能存在的有效性威胁与局限性. 最后一节对全文工作进行总结, 并展望未来的研究方向.

1 研究背景及相关工作

1.1 结构分析和设计语言-AADL

AADL 是由美国汽车工程师学会 (Society of Automotive Engineers, SAE) 标准化的一种专门用于嵌入式系统建模与分析的体系结构描述语言. 该语言采用基于组件 (component-based) 的建模方式, 能够统一描述软件和硬件构件及其交互关系, 通过定义各类系统组件及其接口、实现和属性, 构建系统的抽象模型. 在构件层级上, AADL 定义了 10 类标准组件类型及抽象的包类型. 软件组件作为最顶层的架构容器, 其下可递归包含处理器、进程、线程、子程序等执行单元, 进程也可递归包含线程, 同样硬件组件也包含处理器、设备等. 在实际任务中, 系统中所

涉及的组件类型通常为上述全集的子集. 这种统一的软件-硬件协同建模机制使得 AADL 在嵌入式系统开发中极具表达力, 特别适用于处理任务调度、执行时序、资源分配和跨平台集成等关键设计问题.

在实际系统设计中, 具体项目所涉及的组件类型通常为上述全集的一个子集. 这种统一的软件-硬件协同建模机制, 使得 AADL 在嵌入式系统开发中具有极强的表达力, 尤其适用于任务调度、执行时序、资源分配以及跨平台集成等关键设计问题. 此外, AADL 具备良好的可扩展性, 通过“Annex”机制可进一步定义组件的行为描述、故障传播模型以及对 ARINC653 标准的支持等扩展语义, 从而增强模型在特定应用场景中的适应能力.

1.2 需求文档的结构化分析与条目提取

现代系统日益复杂, 主要由硬件、电子、软件与用户之间日益紧密的交互所驱动. 系统设计往往涉及跨多个组织的协同工作, 因此对高效的数据交换提出了更高要求[12]. 在这一背景下, 需求工程 (requirements engineering, RE) 已成为系统开发过程中的一个关键挑战领域, 对系统开发的成功具有决定性影响[13]. RE 的核心任务在于捕获、表达并管理各类利益相关者的系统需求, 其中自然语言文档作为需求表达的主要形式, 因其可读性强、通用性高, 广泛应用于工程实践中. 然而, 随着需求规模的不断扩大, 这些自然语言编写的需求文档在处理过程中逐渐暴露出一系列问题, 如语义歧义、内容冗余和逻辑不一致等[14], 对后续的系统分析与建模构成挑战. 为解决上述问题, 信息抽取 (information extraction, IE) 的目标是从非结构化或半结构化数据中捕捉和提取有用信息. 在文本数据方面, 信息抽取被用于丰富知识库, 并通过自然语言处理技术对文本进行分析. 现有 IE 方法大致分为两类: 一是基于规则的方法, 该方法依赖领域专家编写抽取规则, 适用于结构清晰、语义稳定的场景, 但在面对语言多样性和复杂表达时, 往往表现出较差的鲁棒性; 二是基于学习的方法, 尤其是近年来随着深度学习和大语言模型的发展, 基于学习的方法在抽取准确性、泛化能力和适应性方面展现出显著优势[15].

尽管 IE 技术已广泛应用于通用文本处理任务, 但在标准化工程文档这一特定领域中, 尚缺乏系统化的解决方案. 为填补这一研究空白, Luttmer 等人[15]针对当前缺乏系统化解决方案从标准文档中自动抽取需求条目并结构化导入需求管理工具的问题, 系统评估了包括规则方法、支持向量机、微调 BERT 在内的 6 种信息抽取技术, 证明了自动需求条目抽取的可行性和有效性. 针对需求粘连问题, Zhao 等人[9]提出了一种名为 DRIP 的两阶段自动化方法, 用于将需求文档中段落级别的粘连式需求划分为独立的单条需求项. 该方法首先通过基于 BERT 和 Siamese 网络的 RS-Siamese 框架学习句间语义关联性进行初步分割, 随后利用基于语义角色标注的最小语义构成规则 (MSC) 对分割结果进行优化, 有效提升了在不同复杂度文档上的分割准确性. 然而, 该方法在句间关系建模和“一句话多个需求”的处理能力上存在局限, 难以准确解构工业文档中常见的复合表达, 进而可能导致 AADL 建模中组件粒度划分失真. Wein 等人[11]提出一种三阶段自动化方法, 首先通过基于动词原形和关键字的句子抽取方式, 从设计文档中识别潜在需求语句; 然后结合句法分析进行指代消解, 通过附加前一句的主语来消除语义歧义; 最后, 采用 TF-IDF 余弦相似度将抽取语句与 DOORS 系统中的正式需求进行匹配, 实现需求对齐与覆盖性验证. 尽管该方法在大规模需求比对中表现出较高的召回率, 但其精度依赖于上下文的有效建模, 运行效率受限于比对规模, 且在复杂语义表达下仍存在一定误判.

除标准文档外, 许多研究者也从其他类型的文本数据中识别和抽取需求, 例如, Akay 等人[16,17]研究了应用不同方法进行基于自然语言处理和 BERT 预训练模型从工程设计文档中抽取结构化的功能性需求. 其研究强调了语义理解在需求抽取中的核心作用, 但也指出, 在面对文本中大量非标准术语、隐喻性表达以及上下文依赖问题时, 模型性能仍存在显著波动. 针对文档理解中涉及图像、文本与布局结构三者融合所导致的高训练与部署成本问题, Fujitake 等人[18]提出了一种将这些多模态信息与 LLM 集成的方法. 该方法充分利用已有文档图像理解研究的成果与 LLM 在语言理解方面的强大能力, 通过在多模态指令数据集上进行微调, 使得一个单一模型即可实现对文档图像的综合理解. 此外, Zhang 等人[19]提出样本中心的上下文学习方法 (SAIL), 通过引入实体级文本相似性和布局相似性, 提升 LLM 对视觉文档中内容与结构的理解. 这里的视觉文档不仅包含文本, 还包含复杂排版、表格、图表和段落结构等视觉元素.

当前面向文档信息抽取的研究虽然取得了一定进展, 但在应对结构多样、语义复杂的工业文档时仍面临着多重挑战. 首先, 大量现有方法依赖基于规则或模板驱动的抽取机制, 对输入文档的结构和格式要求较高, 在面对现

实中格式不统一、语义表达多样化的自然语言文档时, 难以保持抽取粒度的一致性与准确性, 从而限制了其在如 AADL 自动建模等对粒度控制和语义一致性要求较高的任务中的适用性. 其次, 即便部分方法在标准数据集上展现出良好性能, 其对跨领域与非结构化文档的泛化能力不足, 难以应对实际嵌入式系统设计文档中存在的多种句式、交叉表达和上下文依赖. 最后, 近年来涌现的依赖大语言模型或多模态输入的系统, 虽具备较强表达能力, 却通常伴随较高的训练与部署成本, 在资源受限的工业现场或边缘设备中应用仍存在实际障碍.

1.3 AADL 架构生成

AADL 作为一种支持嵌入式系统建模与分析的架构描述语言, 近年来逐渐被广泛应用于系统设计阶段, 特别是在需求建模到系统实现之间起到了桥梁作用. 然而, 由于自然语言需求本身存在语义模糊、结构不规范、上下文依赖强等特点, 直接将其转化为形式化的 AADL 架构模型面临较大挑战. 对此, 学术界提出了多种自动建模技术路径, 涵盖需求模板转换、代码逆向建模以及形式化方法集成等方向, 力图提高从需求到模型的自动化程度与建模质量.

在自然语言模板驱动的 AADL 建模方法中, 王飞等人[20]提出一种基于限定自然语言需求模板和需求抽象语法图的自动化转换方法, 将结构化需求自动转化为 AADL 设计模型. 该方法解决了嵌入式软件开发中自然语言需求难以自动处理以及需求与设计之间缺乏一致性和可追踪性的问题. 然而, 其生成的 AADL 模型主要为静态结构模型, 仍需开发人员在后续阶段手动补全组件行为、数据交互等详细建模信息, 自动化程度仍有限. 类似的, 刘承威等人[21]提出一种基于限定自然语言需求模板 (RNLreq) 和自动转换方法 (RNL2AADL), 将结构化自然语言需求自动转化为 AADL 模型并支持 AGREE 组合验证. 该方法解决了自然语言需求与形式化模型驱动开发之间难以衔接、需求模糊性强及缺乏验证支持的问题, 提升了安全关键软件的可靠性与可验证性. 然而, 该方法对自然语言输入的格式要求严格, 需严格遵循预设模板, 导致其在工业应用中的可扩展性和灵活性较差, 难以适应实际需求表达中的多样性与模糊性.

针对工程实践中存在的设计信息深度嵌入代码、架构层难以还原等方面的问题, 邱志凯等人[22]提出了一种名为 C2AADL 的方法, 即一种从 C 代码自动构造 AADL 模型的逆向工程流程. 该方法系统分析了源代码的结构、行为及运行时特性, 并提出了与 AADL 建模语言之间的转换规则, 覆盖了 AADL 中的组件结构、多线程行为、调度与同步机制等关键元素. 考虑到安全关键领域广泛使用 MISRA C 规范, C2AADL 方法设计了解析策略以确保转换后模型符合安全编码约束. 同时, 作者开发了原型工具并基于实际雷达处理系统进行实验验证, 表明该方法在多核嵌入式平台上具备良好的实用性. 此外, 王日磊等人[23]提出了一种将形式化 B 方法表示的系统需求自动转换为 AADL 架构模型的方法. 其核心思想是: 通过在需求阶段采用形式化方法来确保系统模型的正确性, 并借助 B 方法与 AADL 语言之间的语法映射规则, 实现从需求到架构模型的系统化转换, 从而提升整个系统研制流程中建模的严谨性与测试的自动化程度. 然而, 该方法主要面向已形式化表达的需求, 如何将非形式化需求转化为 B 方法描述并进一步实现自动化转换, 仍是工程落地过程中的关键瓶颈.

尽管已有研究从结构化自然语言、源代码逆向工程以及形式化建模[24,25]等技术路径出发, 提出了多种面向系统架构自动生成的建模方法, 并在一定程度上缓解了自然语言需求难以建模、架构信息易丢失以及设计与验证过程割裂等问题, 在模型驱动开发 (model-driven development, MDD) 框架下展现出良好的工程适用性与发展潜力, 但在实际应用中仍面临若干关键挑战. 首先, 现有方法多数依赖预定义的受限自然语言模板或特定语法结构 (如 RNLreq), 虽然提升了语义解析的准确性和转换规则的可控性, 却对输入需求文本的表达方式提出了严格要求. 这种强依赖限制了方法在处理实际工程中风格多样、语义复杂且存在模糊性的自然语言需求时的适应能力, 难以满足真实项目中通用性和灵活性的要求. 其次, 当前的自动建模流程普遍仍处于“半自动”或“工具辅助”的阶段. 尽管已有方法能够实现从自然语言需求到初步系统模型的自动转换, 但在组件类型识别、接口抽取、调度约束建模等核心环节, 仍需开发人员依据上下文进行人工干预与手动补全, 导致自动化水平有限. 此外, 许多生成的系统工程模型, 尤其 AADL 模型侧重于结构建模, 缺乏对系统行为语义的全面表达, 未能涵盖任务间交互、调度策略、运行模式等关键系统层级. 这种建模不完整性进一步限制了模型在形式化验证、性能分析以及系统集成等后续开发环节中的直接可用性.

2 研究动机

AADL 在描述复杂嵌入式系统结构时工作量大、流程繁琐, LLM 的引入为 AADL 模型的自动化生成提供了新的解决路径, 展现出提升建模效率与质量的潜力. 虽然 LLM 具备较强的代码生成能力, 但由于训练时知识库中缺乏足够的 AADL 相关专业知识, 导致其生成的 AADL 代码语法错误较多, 质量不稳定. 此外, 在系统整体建模层面, LLM 同样表现出严重不足, 具体表现为生成的组件缺乏必要的连接关系, 系统层级结构缺失, 接口命名和语义不一致, 导致模型结构割裂, 难以满足后续分析和验证需求. 以 AADLib 中的 LAMP 时序一致性验证任务为例, 其系统行为包含周期任务调度、状态转换控制以及运行时的执行时间与端口值校验. 在该任务中, 某一线程 Initial_State 的作用是, 在每个周期开始时初始化输入变量并记录起始时间, 为后续状态处理提供数据与时间基准. 该线程应执行以下操作: 依次调用两个子程序: assign_random_values 与 record_start_time; 在操作完成后通过事件端口 state_change 触发状态转移; 定义线程的计算执行时间区间为 2–4 ms. 采用 DeepSeek-r1 模型读取该功能需求描述后生成图 1 所示的 AADL 代码.

```
1 :   thread Initial_State
2 :   end Initial_State;
3 :   thread implementation Initial_State.Impl
4 :   calls
5 :       sequence: {
6 :          assign_random_values,
7 :          record_start_time
8 :       };
9 :   features
10:       state_change: out event port;
11:   properties
12:       Compute_Execution_Time => 2 ms . 4 ms;
13:   end Initial_State.Impl;
```

图 1 DeepSeek-r1 直接生成 LAMP 时序一致性验证 AADL 代码示例

该代码在结构上表达了正确的意图: 先后调用两个函数, 并通过事件端口触发状态转移. 然而, 经语义与结构分析后, 发现其在多个关键位置存在严重建模错误如下.

1) 第 9、10 行 features 应定义在线程类型 (Thread) 中, 而非实现体 (thread implementation) 中. 该错误将导致模型在语义检查阶段失败, 无法建立端口接口.

2) 第 4–8 行中的 calls 部分未正确声明所调用子程序类型与路径. AADL 规范要求使用 Subprogram 引用, 如图 2 所示.

```
1:   calls myseq: {
2:       function1: subprogram assign_random_values.Impl;
3:       function2: subprogram record_start_time.Impl;
4:   };
```

图 2 Subprogram 引用示例

当前语法缺失命名标识符 (如 myseq、init) 和 Subprogram 类型标注, 不能被调度器正确识别, 该语法错误也是目前大多数 LLM 频繁发生的错误之一. 此外, 在对 LAMP 架构整体的生成过程中还发现, 从调度关系角度来看, LLM 在生成线程调用子程序以执行具体功能时, 经常遗漏与之对应的组件声明. 这一缺失直接导致调度链条不完整, 进而影响系统的执行路径解析与任务调度配置. 在功能实现层面, 生成模型普遍忽略了周期性任务调度的基本需求, 未能为关键线程设置周期属性, 导致模型缺乏实时性保证与调度稳定性. 更进一步地, 生成模型往往将“初始化”“变量计算”和“周期输出”分别建模为 3 个独立状态, 并通过线程间的事件触发实现状态迁移. 虽然这一思路在功能逻辑上具备一定合理性, 但其设计过程中忽视了 AADL 中组件 (component) 与模态 (mode) 在语义上的根本差异: 前者用于定义结构与功能单元, 后者用于建模行为的切换关系. 混淆两者语义将导致状态转换逻辑不符合 AADL 标准, 破坏模型的可验证性与可部署性. 这一段代码的问题并非孤立现象. 在多个复杂任务中, 直接生成结果普遍表现出行为逻辑与架构结构相脱节、接口定义不规范、调度约束缺失等共性缺陷. 这表明现有 LLM 模型

尚未具备处理复杂嵌入式建模语言所需的结构理解与语义推理能力, 特别是在结构化高, 具有强依赖关系和调度约束的建模语言中, 端到端生成策略难以保证系统可验证性与工程可用性.

3 数据集构建

本文选取 AADLib 库以及 AADL Inspector 官方发布的典型示例作为构建数据集的基础资源 (https://github.com/OpenAADL/AADLib). AADLib 是一个具备良好复用性和广泛工具支持的 AADLv2 标准模型库, 覆盖多种典型的嵌入式系统构件, 具备丰富的建模元素与属性配置, 提供了完善的 AADL 构件与系统实例, 但其缺乏与自然语言需求之间的一一对应关系, 限制了其在需求驱动的建模任务中的应用价值. 为实现从形式化架构到语义表达的高质量转换, 本文设计了如图 3 所示的流程, 引入两个智能体协同工作: 第 1 个智能体负责解析 AADL 中的组件结构与调度语义, 生成粒度一致的自然语言功能描述; 第 2 个智能体进一步对组件级表述进行聚合与组织, 生成具备层级结构的系统架构文档. 此种划分旨在分离结构解析与语义生成过程, 提升生成文本的对齐性、可读性与系统一致性, 为建模提供可靠的数据支撑.

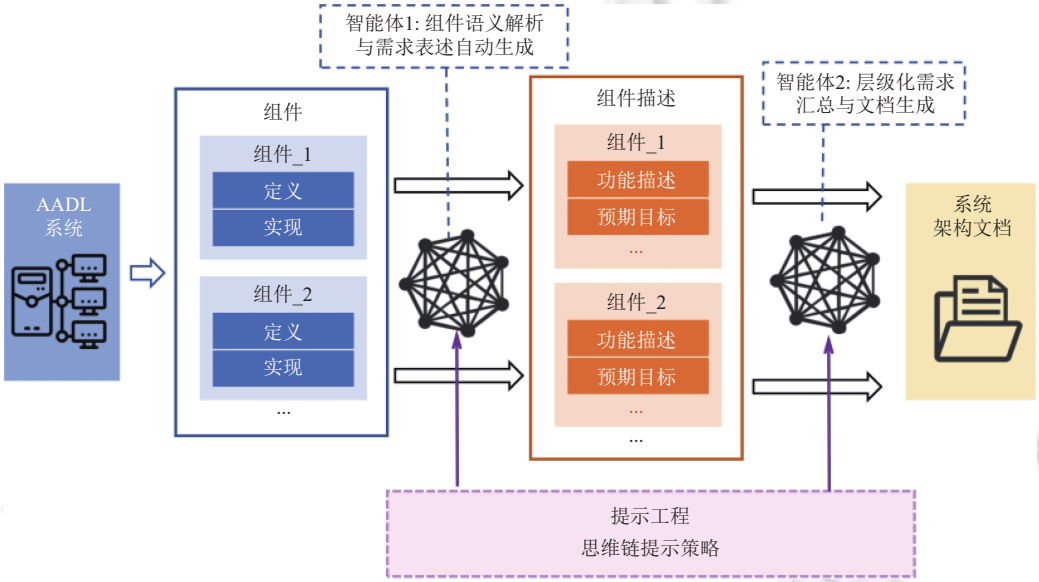


图 3 AADL 架构到系统架构文档生成原理图

3.1 AADL 组件语义解析与需求表述自动生成

本文首先对 AADLib 库中提供的系统模型进行了分析, 选取了不同复杂度和规模的典型系统实例. 在第 1 阶段, 设组件代码集合为 $C=\{c_1,c_2,\ldots,c_n\}$, 每个组件 c_i 被直接输入 LLM, 生成对应的自然语言功能描述 d_i:

$$d_i = LLM(c_i),\ \forall c_i \in C \tag{1}$$

在此基础上, 本研究针对提示工程中的问题提出一种基于思维链提示 (chain-of-thought (CoT) prompting) 策略的引导范式, 用于逐步驱动模型识别、解析并转换 AADL 组件. 该策略通过引导 LLM 依次完成组件注释识别、类型与接口解析、实现特征分析以及需求生成等步骤, 实现对组件功能、接口交互及运行行为的系统化理解与表达, 从而避免信息割裂与推理中断的问题, 确保生成的需求描述具有完整性与一致性. 由此构建出组件描述集合 $D=\{d_1,d_2,\ldots,d_n\}$, 该集可表达各组件的功能职责、接口交互与行为特征, 也作为系统级生成的中间语义载体.

本节所提出的 AADL 组件生成条目化需求的方法需要在两方面进行应用, 包括支撑系统架构文档数据集生成以及 AADL 组件知识库构建. 为了避免知识库中包含实验数据的情况, 两部分的数据集分属不同的 AADL 架构. 本文基于 AADLib 及 AADL Inspector 官方给出的相关典型模型完成包含超过 500 个组件的知识库构建, 数据集构成如表 1 所示, 其中组件定义 461 个, 组件实现 259 个, 涵盖软件组件、硬件组件及复合系统类型组件, 形成

了具有代表性的语义样本集合. 该样本集合被用作后续检索增强生成 (retrieval-augmented generation, RAG) 的知识库. 该知识库以表格或 JSON 格式组织, 系统记录每个组件的类别、接口信息、运行时属性及功能语义, 结合链式提示策略生成的自然语言需求描述, 并通过专家验证确保其准确性, 实现了语义结构与需求描述的紧密对应与双向映射.

表 1 AADL 组件知识库信息统计

组件	类型	计数	组件	类型	计数
应用软件	线程	162	硬件	处理器/虚拟处理器	31
	进程	64		总线/虚拟总线	9
	子程序	58		内存	14
	数据	26		设备	63
	抽象	8	复合组件/系统	系统	67

3.2 层级化需求汇总与文档生成

针对 AADL 架构-系统架构文档数据集的构建, 在完成基于 LLM 的组件级语义解析后, 为实现从原子级建模元素到系统层级需求的结构化组织, 在第 2 阶段, 本文将上述组件描述集合 D 整体输入至大语言模型, 引导其基于多个组件间的功能关联、通信关系与层次结构, 生成系统层级的自然语言系统架构文档 S:

$$S = LLM(d_1, d_2, \ldots, d_n) \tag{2}$$

为支持系统概述内容的自动化生成, 本文设计了面向 LLM 的提示模板, 结合功能归类; 组件交互推理与分层结构建模等关键技术, 引导模型生成符合系统工程规范的系统架构文档.

本阶段最终构建的数据集包含 3 类主要元素.

- AADL 原始组件代码 c_i, 作为系统建模的基础语义单元, 体现具体组件的功能定义与属性描述.
- 组件级自然语言描述 d_i, 由 LLM 基于 AADL 代码生成, 涵盖每个组件的功能说明、通信接口与交互逻辑.
- 系统级自然语言系统架构文档 S, 由多个组件描述融合生成, 完整反映系统结构、模块划分、功能集成与交互关系, 是支撑系统工程活动的高层级文本表达形式.

AADLib (49 个项目) 和 AADL Inspector examples (44 个项目) 作为数据来源, 其中大部分项目用于知识库构建, 同时从两个数据源中共选取 15 个完整项目作为独立测试集, 具体测试集选择见表 2.

表 2 实验架构数据统计 (个)

数据来源	系统	名称	处理器	进程	线程	子程序	组件实现数量	子组件数量	特征数量	连接数量
AADLib	时间触发式嵌入式系统架构	Time_trigger	1	1	3	3	8	5	103	6
	生产者消费者嵌入式系统	VxWorks	2	2	2	2	5	9	101	7
	雷达信号处理系统	Radar	1	1	5	4	8	14	127	25
	Car分布式实时控制系统	Car	3	3	9	0	7	16	109	10
	ROSACE系统	ROSACE	1	1	12	12	33	44	205	110
	Ardupilot飞行控制系统	Ardupilot	2	3	3	6	24	80	155	125
AADL Inspector examples	心脏起搏器	Pacemaker	1	1	3	0	4	8	120	15
	空调控制系统	Air_conditioner	1	1	3	1	7	11	112	15
	卫星控制平台	Satellite	1	1	14	0	2	26	163	43
	太阳能汽车	Solar_car	3	3	7	0	30	28	166	59
	冗余采集	Redundancy	1	1	4	4	10	18	116	18
	总线通信的端到端控制系统	End_to_end	3	3	5	0	19	42	157	46
	实时周期性任务调度	Periodic_task	1	1	2	4	10	8	95	0
	LAMP在线分析系统	LAMP	1	1	3	3	4	10	103	4
	“传感-处理-执行”控制系统	Control_sys	4	6	0	0	10	28	174	65

4 SmartGen-AADL 方法论

针对当前自然语言需求表达中普遍存在的多源异构, 即需求来源多样且表达形式差异明显以及语义隐含等问题, 本文引入多智能体协同机制, 提出一种面向复杂系统建模的多智能体语义解析方法 SmartGen-AADL. 该方法围绕自然语言需求的结构化理解、语义解析与模型生成这 3 个阶段展开, 构建了由多个智能体组成的顺序执行、信息单向传递的协同工作流程, 整体框架如图 4 所示. 在具体设计上, 方法由 3 个核心模块构成. 首先, 系统架构文档结构化模块部署两个智能体, 分别负责从自然语言文本中提取系统架构信息与生成标准化的句级需求条目, 解决表述冗余、层级不清等问题, 其标准化输出将作为下一阶段智能体的输入. 其次, 子问题分析模块的智能体接收上述结构化条目对结构化条目进行语义细化, 用于识别子问题并自动挖掘模块间的交互关系与接口定义, 从而增强需求信息的完整性与可建模性. 最后, AADL 组件生成模块的智能体基于前序模块输出的精细化语义信息, 结合 LLM 的生成能力与知识库的上下文检索机制, 自动构建面向任务的 AADL 组件及其连接关系, 形成从语义理解到架构建模的闭环自动化流程.

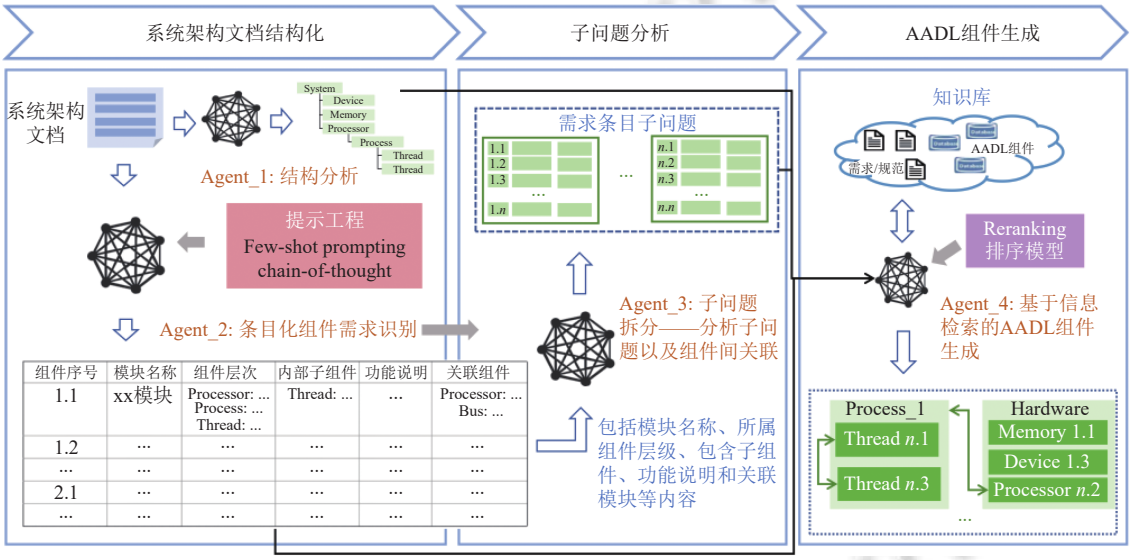


图 4 SmartGen-AADL 方法框架图

4.1 系统架构文档结构化模块

为了实现从自然语言系统文档到 AADL 建模的自动转化, 本模块首先设计了面向组件识别与层次解析的结构化流程, 旨在明确系统中应包含的各类组件及其组合结构. 该流程以 AADL 基础构件类型为建模单元, 结合对系统架构设计文档的语义分析, 识别其中所蕴含的功能模块、执行实体及其嵌套组织方式. 本模块中设计了提示模板, 使大语言模型能够以链式思维方式进行语义驱动的解析. 整个流程包括两个主要阶段: (1) 结构识别阶段, 用于提取文档中的系统分层结构及执行单元边界; (2) 条目提取阶段, 用于细化每个功能模块内部的最小可执行功能单元. 通过上述提示配置, 模型能够对复杂段落进行语义分解, 提取模块职责与关联组件, 并构建出如下树状结构的系统实例层级.

以一个具体的实例来看, 针对“飞机高度控制系统”的示例需求, 其组件结构层级可表示为系统树. 完整系统层次结构包含气压传感器、计算处理器、显示器、控制任务线程等多个子模块, 涵盖 Device、Processor、Thread、Subprogram 等多类 AADL 构件. 输入第 1 个智能体的原始需求文本内容为: (1) 气压传感器获取外界气压后, 通过 PCIe 总线传递至高度计算分区的高度计算任务; (2) 高度计算任务调用高度计算程序, 计算出飞机高度后, 通过 PCIe 总线传递到显示器; (3) 空速控制指令传感器获取高度控制指令后, 通过 PCIe 总线传递至高度响应分区的

高度响应任务; (4) 空速响应任务接收指令后, 通过 3 个 PCIe 总线向油门、升降舵和副翼发送控制指令, 完成飞机高度控制. 生成的系统架构图示例如图 5 所示.

```
System: 飞机高度控制系统 (Height_Control_System)
├── Device: 气压传感器 (Pressure_Sensor)
├── Device: 空速控制指令传感器 (Control_Command_Sensor)
├── Processor: 高度计算分区 (Height_Computation_Processor)
│   └── Process: 高度计算任务 (Height_Calculation_Process)
│       └── Thread: 高度计算线程 (Height_Calculation_Thread)
│           └── Subprogram: 高度计算程序1 (Altitude_Calc_Program)
├── Processor: 高度响应分区 (Height_Response_Processor)
│   └── Process: 高度响应任务 (Height_Response_Process)
│       └── Thread: 高度响应线程 (Height_Response_Thread)
├── Device: 显示器 (Display_Device)
├── Device: 油门控制单元 (Throttle_Controller)
├── Device: 升降舵控制单元 (Elevator_Controller)
├── Device: 副翼控制单元 (Aileron_Controller)
├── Bus: PCIe总线 (PCIe_Bus_1, PCIe_Bus_2, PCIe_Bus_3)
```

图 5　飞机高度控制系统架构图示例

上述系统架构图展示了“飞机高度控制系统”在需求层驱动下的模块化分解与构件组织方式. 通过语义解析与结构识别, 需求中的功能语句被映射为对应的系统组件及其交互关系. 其中, 传感器类 Device 负责外部环境与控制信号的感知输入, Processor 及其子层级组件承担核心计算与任务调度功能, 而多个 PCIe 总线构件则作为关键通信通道, 支持不同模块间的数据传输.

在结构分析的基础上, 进一步将系统架构文档按章节与功能模块的方式拆分, 每个分块可以视为一个相对完整的逻辑功能模块的描述单元. 为进一步规范每个组件的结构化描述, 本文将其统一抽象为五元组的形式:

$$c_i = (Module_i, Subcomponents_i, Function_i, Relation_i) \tag{3}$$

其中, $Module_i$ 表示组件名称, $Subcomponents_i$ 表示该组件内包含的子模块集合, $Function_i$ 表示组件的功能说明, $Relation_i$ 表示该组件与其他模块间的连接或交互关系. 为了系统化提取架构文档中的结构信息, 本文设计了一套基于 LLM 的条目化需求识别流程. 该流程结合提示工程方法, 通过结构化提示模板与迭代式交互, 引导模型对原始系统文档进行结构解析与属性抽取. 整体流程包括章节结构识别、组件对应关系确认、组件属性分析、连接关系提取与模块化组织这 5 个阶段, 逐步将自然语言文档映射为具备工程语义的条目化结构描述. 为保证识别结果的完整性与规范性, 提示模板明确限定了分析逻辑、思维框架与语法要求, 并以真实系统的识别结果作为示例, 辅助模型在理解、提取与表达方面达到结构对齐目标.

4.2　需求子问题分析模块

由于当前 LLM 在建模语言语法掌握不系统、上下文约束能力有限以及对系统工程语义理解不足等问题仍然存在, 导致其在直接生成 AADL 架构模型时准确性较低, 常出现接口不一致、属性遗漏或结构混乱等现象, 难以满足建模规范与形式化验证的要求. 为提升建模质量与生成准确性, 有必要在组件生成之前引入对原始需求的进一步结构化分析. 智能体 2 的条目分析之后, 中间结果已具备初步映射为 AADL 组件的能力. 然而, 为了构建一个完整、结构合理且符合系统工程规范的 AADL 模型, 仅依赖基本的组件类型判定仍远远不够.

AADL 的建模机制包含组件定义 (component type) 与组件实现 (component implementation) 两个层次, 前者主要描述组件的外部接口与属性, 后者则负责建模组件的内部结构、子组件构成及其连接关系. 其中, 组件定义侧重于接口声明与属性设置, 而组件实现则描述了内部结构、子组件组成及连接方式等. 为了实现对这两部分的生成, 需进一步识别以下关键信息: 组件的输入输出端口类型与方向、调用关系、调度属性 (如执行周期、优先级、处理能力) 以及资源约束等. 这些信息部分在原始需求中显式呈现, 而更多则隐含于上下文、句法关系或语义指代之

中, 必须借助进一步的分析与推理进行补全. 为此, 本文引入了第 3 阶段的子问题拆分 (智能体 3), 其任务是对条目化需求进行子问题级分析, 从而支持组件级 AADL 模型的精细生成. 其具备以下核心能力: 将复合需求细化为可映射到 AADL 特征或属性的子问题 (需求分解), 补全隐含但关键的接口和约束信息 (语义补全), 识别组件交互方式如端口和调用接口 (特征提取), 提取周期、优先级等非功能信息 (属性识别), 为精准生成 AADL 模型提供语义支撑. 图 6 和图 7 中的实例展示了基于上述提示模板对条目化需求的细化分析过程.

```
<background>
你是一名精通AADL语言的软件工程师,正在根据系统架构文档和组件结构,分析其对应AADL项目的组件细节.
</background>
<adaptive_thinking_framework>
你应当严格遵循以下思考步骤进行分析: - 整体概述:明确文档对应的项目 - 章节排布:明确各章节之间的关系,(例如:按照功能分章、按照组件类型分章或其他) - 确认对应关系:根据组件树中的每个组件,确定其在原文章对应的段落(也许不止一处) - 分析组件属性:根据上一步骤确定的一系列段落内容,记录每个组件的持有的属性信息 - 分析组件连接关系:分析组件间的连接关系(绑定、包含、连接等) - 模块组织:通过若干功能模块组织上述信息
</adaptive_thinking_framework>
<instruction>
你应当遵从以下指令,以优化结果:
- 对于每个组件都应当详细检查原需求文件内容,*必须*确保属性没有遗漏 - *禁止*自由发挥,构建不存在的属性 - 连接关系*严格遵守*AS5506C语法规范,不能构建不合法的连接方式(例如process与processor应该在system绑定,而不是在其他环节)
</instruction>
以下是基于前述飞行高度控制系统的结果示例:
```

```
"飞行控制执行模块": {
  "组件层次": ["Throttle_Controller", "Device: Elevator_Controller", "Device: Aileron_Controller"],
  "内部子组件": [],
  "功能说明": "接收来自“高度响应任务模块”的控制信号.油门:控制发动机推力;副翼:稳定飞行方向.",
  "关联组件": ["Processor: Height_Response_Processor", "Bus: PCIe_Bus_3"] },
"高度计算任务模块": {
  "组件层次": [ "Processor: Height_Computation_Processor",
    "Process: Height_Calculation_Process",
    "Thread: Height_Calculation_Thread",
    "Subprogram: Altitude_Calc_Program" ],
  "内部子组件": ["Subprogram: Altitude_Calc_Program"],
  "功能说明": "接收由气压传感器通过PCIe总线传来的气压数据.使用气压值结合本地已存储的校准参数进行高度计算.计算方法:解析气压数据,转换为海拔.高度值计算完成后,立即通过PCIe总线发送至显示器,用于飞行状态监控.",
  "关联组件": ["Device: Pressure_Sensor",
    "Device: Display_Device",
    "Bus: PCIe_Bus_1(接收气压)",
    "Bus: PCIe_Bus_2(发送高度)"] }
```

图 6 系统架构文档结构化模块提示模板

输入-条目化需求:	输出-细化分析结果
"组件": "CAN_Bus", "对应功能": "作为系统通信骨干,连接所有电子控制单元;支持全双工PUSH型数据传输;固定延迟1毫秒,字节级传输延迟1微秒", "子组件": "无", "关联关系": ["System: Dashboard", "System: Motors", "Device: SolarCells", "Device: Batteries", "System: Signalling", "System: MainECU"]	以示例BUS_CAN_others.impl为参考 .具体分析如下: 1. 总线实现属性(Bus Implementation Properties) 总线实现 BUS_CAN_others.impl 定义了多个关键属性: Data_Transfer_Type 设置为 Full_Duplex(全双工),表示数据可以同时进行发送和接收; Transmission_Type 设置为 Push(推送),意味着数据主动发送,无需接收方请求; Fixed_Latency 设置为 1 毫秒,表示每次传输存在固定的延迟; Byte_Level_Latency 设置为 1 微秒,表示每个字节数据的传输延迟 . 2. 连接绑定(Connection Binding) 连接 Connection_Bus 将源端口 Thread_Source_Port.out 绑定到目标端口 Thread_Destination_Port.in,该连接通过 BUS_CAN_others 总线实现,确保通信过程符合其定义的属性要求 . 3. 属性集定义(Property Set Definition) 属性集 Communication_Properties 定义了用于总线实现的属性类型: Data_Transfer_Type 是一个枚举类型,其可能取值包括 Full_Duplex(全双工)和 Half_Duplex(半双工); Transmission_Type 也是枚举类型,可能取值包括 Push(推送)和 Pull(拉取); Fixed_Latency 与 Byte_Level_Latency 均为 Time 类型,用于描述通信的时延特性 . 总结:总线实现BUS_CAN_others.impl支持全双工传输方式和基于推送的数据传输机制,通信延迟包括固定延迟1毫秒和每字节传输延迟1微秒.连接Connection_Bus通过该总线实现线程端口间的数据通信,确保通信过程遵循已定义的属性规范.这些属性均属于Communication_Properties属性集,为系统提供了正式且可配置的通信行为建模能力.

图 7 对条目化需求的细化分析过程实例

通过对条目化需求的系统性细化分析, 实现了对组件接口、交互关系及调度配置等关键要素的精准提取, 为后续符合 AADL 规范的模型自动生成奠定了语义支撑与结构基础.

4.3 AADL 组件生成模块

4.3.1 AADL 知识库构建

在前述步骤中生成的结构化需求内容, 旨在作为应用 LLM 进行 AADL 组件自动生成的输入信息. 经过语义

解析与子问题识别后, 系统已形成可形式化表示为模块集合的输入结构:

$$M = \{M_1, M_2, \ldots, M_n\} \tag{4}$$

其中, 每个模块 M_i 包含相应的功能描述、接口定义、属性约束及与其他组件的关联关系. 基于上述输入形式, 为实现高质量的 AADL 构件生成, 还需克服模型知识不足带来的挑战. 为此, 本文引入了基于 RAG 的技术框架. 首先设计并构建了一个 AADL 领域知识库, 如图 8 所示, 该知识库由两部分构成, 分别对应语言规范层的建模规则与实例层的语义映射.

$$K_{\text{AADL}} = K_{\text{rules}} \cup K_{\text{examples}} \tag{5}$$

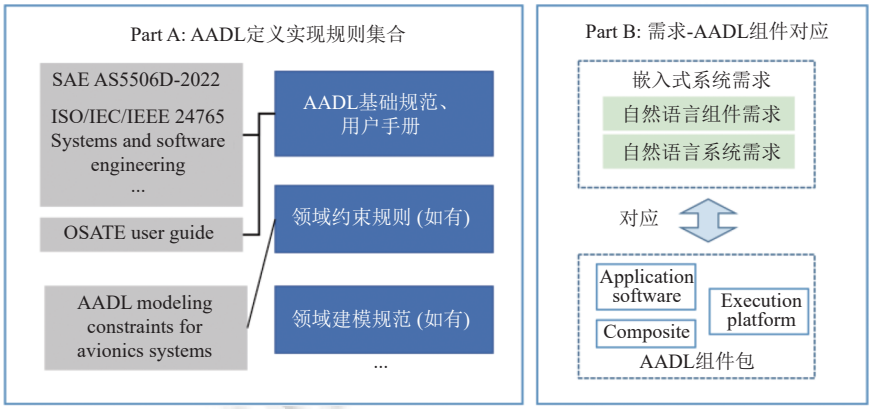


图 8 AADL 知识库架构

● Part A, K_{rules} 为规则集与语义规范部分: 包括 AADL 语言的基础语法规范、组件定义规则、连接机制、线程调度策略、Annex 扩展等官方文档内容, 如 SAE AADL 标准规范与用户手册. 同时结合特定行业 (如航空、航天、轨道交通) 中的建模约束和架构设计经验, 形成领域特定建模规则集.

● Part B, K_{examples} 为需求-组件映射样例集: 汇集结构化需求与 AADL 组件之间对应关系样例, 涵盖设备、处理器、子程序等典型构件, 记录需求描述、组件层级、交互总线及相关注释, 为 LLM 提供上下文匹配模板.

$$K_{\text{examples}} = \{(d_i, c_i) | d_i \in D, c_i \in C\} \tag{6}$$

其中, D 为需求集合, C 为组件集合. 每一对 (d_i, c_i) 表示将需求 d_i 映射为符合语义约束的组件结构 c_i.

4.3.2 AADL 组件生成

在基于 RAG 的组件生成过程中, 首先依据输入的结构化需求内容, 利用语义向量检索机制从构建好的 AADL 领域知识库中获取候选的建模样例及规则片段. 检索完成后, 系统会对所有候选条目按照语义相似度得分进行排序, 并选取与当前需求最相关的 Top-k 条目. 通过这一“语义相似度排序”步骤, 可以有效过滤噪声候选, 确保仅保留与需求高度相关的知识条目. 随后, 这些排序后的高相关内容被嵌入提示模板中, 用于增强语言模型对领域知识的感知能力, 从而引导其生成符合 AADL 语法规范与建模约束的组件定义. 整个流程由 4 个核心环节协同构成: 系统首先对结构化需求进行编码, 生成语义向量, 并与知识库中的已有实例进行相似度匹配与排序, 以完成高质量的相关信息检索. 整个流程如图 9 所示.

在这个步骤中, 对于每条结构化需求条目 $M_i \in M$, 系统通过语义编码模型将其表示为向量表示:

$$v_i = Encode(M_i) \tag{7}$$

其中, $v_i \in \mathrm{R}^d$ 表示模块 M_i 的语义表示向量, d 为编码维度. 随后, 在构建完成 AADL 领域知识库 K_{AADL} 中, 以该语义向量为检索键, 在规则子库 K_{rules} 和样例子库 K_{examples} 中分别执行语义相似度匹配, 获得与当前条目语义高度相关的建模规则片段与组件定义实例, 本文使用向量检索引擎 FAISS 对两个子库中已编码的语义向量集合进行近邻搜索, 获得与当前模块语义最相关的 k 个结果, 从规则库中检索的前 k 条规则片段集合定义为:

$$R_i^{\text{rules}} = \left\{r_1^{(i)}, r_2^{(i)}, \ldots, r_k^{(i)}\right\},\ r_j^{(i)} \in K_{\text{rules}},\ \forall j \leqslant k \tag{8}$$

其中, 每个 $r_j^{(i)}$ 表示一条语义上与 M_i 高度相似的 AADL 建模规则片段, 例如组件约束语法、连接机制、调度策略等. 对应地, 从组件样例库中获取的前 k 个组件定义与实现片段集合为:

$$R_i^{\text{examples}} = \left\{e_1^{(i)}, e_2^{(i)}, \ldots, e_k^{(i)}\right\},\ e_j^{(i)} \in K_{\text{examples}},\ \forall j \leqslant k \tag{9}$$

其中, 每个检索结果项 $e_j^{(i)}$ 可形式化表示为一个三元组: $e_j^{(i)}=\left[\text{NL}_j^{(i)}, \text{Type}_j^{(i)}, \text{Impl}_j^{(i)}\right]$. $\text{NL}_j^{(i)}$ 表示与该组件样例对应的自然语言需求描述, 用于表达需求的语义上下文; $\text{Type}_j^{(i)}$ 表示该条样例中组件的 AADL 组件定义, 即其特征与接口规范等结构信息; $\text{Impl}_j^{(i)}$ 表示该组件的组件实现. 若当前样例未给出实现细节, 则该字段可为 $\varnothing$.

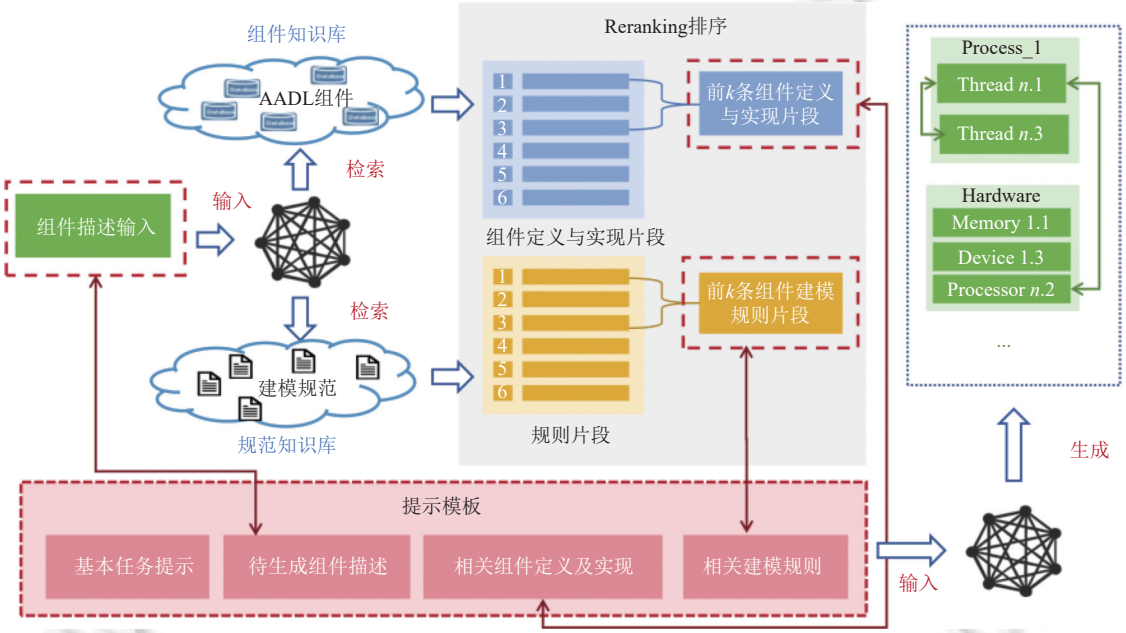


图 9 基于 RAG 的 AADL 组件生成流程

接着, 将检索得到的内容插入至 Prompt 模板中, 形成增强后的输入语境, 进一步为语言模型提供明确的生成引导, 在提示构造阶段, 为确保 LLM 在生成过程中充分感知结构化需求的语义意图与相应的建模规则背景, 本文将原始结构化需求与其对应的知识检索结果共同构成一个增强提示 (augmented prompt), 作为最终输入语言模型的语境上下文. 本文定义增强提示模板如下:

$$Prompt_i = Intro + M_i + R_i^{\text{rules}} + R_i^{\text{examples}} \tag{10}$$

其中, *Intro* 为任务说明或基础问题描述 (例如: “请根据以下信息生成对应的 AADL 组件定义与实现…”), 用于激活语言模型的生成意图. 随后, LLM 在增强提示的基础上生成 AADL 组件定义, 同时配合 AADL 语法解析器进行初步的语义与结构验证, 确保生成结果可被建模工具链接与识别; 最后, 系统将多个结构化需求所对应的组件内容集成组织至系统架构中, 确保生成模型在结构、命名与交互接口等方面保持一致性和完整性, 从而实现从需求驱动到模型生成的自动化闭环.

5 实验设计

5.1 实验数据

为支持从系统架构文档自动生成 AADL 架构模型的研究, 本文基于 AADLib 与 AADL Inspector 官方发布的

典型示例, 构建了一个包含 15 个具代表性系统的实验数据集. 这些系统覆盖了时间触发式嵌入式架构、医疗电子设备、航空航天平台、车载控制系统与工业自动化等多个领域, 体现出较高的应用广度与结构异质性. 本文以各系统提供的 AADL 代码为基础, 结合人工标注与工具辅助的方式, 提取完整架构实现信息, 进一步反向生成系统架构文档与系统架构需求文档, 为自动建模算法构建输入输出对. 为了覆盖多样化的建模需求和架构风格, 本数据集选取的系统在设计目标、功能组织方式与执行机制等方面均存在显著差异. 部分系统强调周期性任务的调度与分析流程, 部分系统则构建了面向多处理单元的分布式控制架构. 表 2 总结了这些系统在建模层面上的关键指标, 从数据分布来看, 特征数量范围在 95–205 之间, 显示出系统在建模粒度和复杂度上的显著差异. 例如, 实时周期性任务调度系统与 LAMP 在线分析系统结构较为紧凑, 功能聚焦, 组件层次较浅; 相比之下, ROSACE 系统与图像处理与数据传输系统则包含更多的子组件与连接关系, 模型规模更大、结构更为复杂 (本文数据以及最终工具已发布在 https://github.com/amethyst6174).

5.2 评价指标及基准模型

为全面评估所提出的基于 AADL 架构自动生成方法的有效性与实用性, 本文设计了 4 类评价指标, 分别从结构对比维度、代码质量维度、语义相似度维度以及人工评价维度进行考察, 确保评估的客观性与全面性.

(1) 构件数量偏差 (component count deviation ratio): 该指标衡量生成架构在组件数量上对原始架构的还原程度. 设原始 AADL 架构包含组件数为 N_{ref}, 生成架构中组件数为 N_{gen}, 构件数量偏差度定义为:

$$C_{\text{dev}} = \frac{\left|N_{\text{gen}} - N_{\text{ref}}\right|}{N_{\text{ref}}} \times 100\% \tag{11}$$

该指标反映生成结果在结构规模上的精度, 值越接近 0 表示组件数量越接近原始设计.

(2) 错误行占比与错误组件占比: 为评估生成模型在代码准确性方面的表现, 本文引入两个互补的指标.

● 错误行占比 (line error rate) 用于度量生成代码中的语法错误的行数占总代码行数的比例:

$$L_{\text{error}} = \frac{L_{\text{err}}}{L_{\text{total}}} \times 100\% \tag{12}$$

其中, L_{err} 表示生成代码中被解析器或人工标注为错误的代码行数, L_{total} 为总生成行数.

● 错误组件占比 (error component ratio) 表示生成组件语义存在错误的模块, 在功能定义、接口连接或组件属性方面与参考架构不一致的组件, 即占总组件数的比例:

$$C_{\text{error}} = \frac{N_{\text{err-comp}}}{N_{\text{gen}}} \times 100\% \tag{13}$$

其中, $N_{\text{err-comp}}$ 为存在语法错误、未封闭定义或结构错误的组件个数.

(3) 架构语义相似度 (semantic similarity): 该指标用于衡量生成架构与人工设计的原始架构在文本层面的语义接近程度. 本文选用 BERTScore[26]作为主要指标. 相比 BLEU[27]和 METEOR 等依赖 n-gram 精确匹配的方法, BERTScore 基于上下文嵌入, 能够更好地捕捉代码结构中的语义等价关系, 避免对词序和表面形式的过度依赖, 特别适合评估结构灵活但语义保持一致的系统架构描述语言. 具体地, 对于每个参考词元 x_i, 在候选句 $\hat{x}$ 中寻找与之最相似的词元, 计算召回率; 对于每个候选词元 $\hat{x}_j$, 在参考句 x 中寻找最相似的词元, 计算精度, 其中使用了贪心匹配策略, 来最大化匹配相似度. 公式如下:

$$R_{\text{BERT}} = \frac{1}{|x|} \sum_{x_i \in x} \max x_i^{\top} \hat{x}_j,\ \hat{x}_j \in \hat{x} \tag{14}$$

$$P_{\text{BERT}} = \frac{1}{|\hat{x}|} \sum_{\hat{x}_j \in \hat{x}} \max x_i^{\top} \hat{x}_j,\ x_i \in x \tag{15}$$

$$F_{\text{BERT}} = 2\frac{P_{\text{BERT}} \cdot R_{\text{BERT}}}{P_{\text{BERT}} + R_{\text{BERT}}} \tag{16}$$

(4) 人工评分 (human evaluation): 尽管自动化评价指标在一定程度上能够量化模型性能, 但在嵌入式系统建模等高要求场景中, 仅依赖结构或文本层次的相似度仍不足以全面判断架构的工程实用性与可部署性. 为进一步评

估模型在工程应用中的可用性, 本文引入人工评审机制, 由具备 AADL 建模经验的领域专家对生成的架构模型进行多维度的主观评分, 以补充自动评价所遗漏的语义与结构判断能力. 本评估共邀请了 3 名具有 AADL 研究经验的研究生以及嵌入式系统背景的专业建模工程师. 评估对象为每轮实验生成的 AADL 系统架构模型, 并从如表 3 所示的 3 个维度对每个样本进行逐项打分.

表 3 人工评分规则

编号	评价维度	说明	分值范围	评分细则
D1	功能完整性	是否全面覆盖需求中指定的所有功能目标, 包括输入、处理和输出链条	1、3、5分	1分: 功能缺失严重, 仅覆盖少数需求目标; 3分: 基本覆盖主要功能, 但存在部分缺失; 5分: 完整覆盖所有功能目标, 无遗漏
D2	逻辑清晰性	架构结构是否合理, 组件划分是否清晰, 通信路径与控制流程是否连贯	1、3、5分	1分: 结构混乱, 组件划分不合理; 3分: 逻辑基本连贯, 但存在不必要的复杂性或局部模糊; 5分: 逻辑完全清晰, 组件职责明确, 通信流程顺畅
D3	接口准确性	各组件间端口、连接机制是否合理, 是否存在端口遗漏或绑定错误等问题	1、3、5分	1分: 接口错误较多, 通信机制严重缺失; 3分: 大部分接口正确, 但存在少量遗漏或错误; 5分: 接口定义完全正确, 无遗漏或冲突

每位专家对每一维度进行打分, 最终得分采用加权平均方式计算. 功能完整性与接口准确性权重较高 (各占 0.35), 逻辑清晰性占比略低 (0.3), 主要考虑前两者直接影响系统正确性与接口可靠性, 而逻辑清晰性虽重要, 但具有一定主观性.

$$H_{\text{score}}^{(i)} = 0.35 \cdot s_{\text{D1}}^{(i)} + 0.3 \cdot s_{\text{D2}}^{(i)} + 0.35 \cdot s_{\text{D3}}^{(i)} \tag{17}$$

其中, $s_{\text{D}x}^{(i)}$ 表示第 i 位专家对第 x 个维度的打分. 最终模型主观评分则由所有专家评分取平均得到:

$$H_{\text{final}} = \frac{1}{n}\sum_{i=1}^{n} H_{\text{score}}^{(i)}, n = 3 \tag{18}$$

5.3 实验流程

本实验所用测试用例主要来源于 AADL Inspector 工具自带示例及开源项目 AADLib, 涵盖通信控制、任务调度、接口管理等 15 个典型应用场景. 为验证所提方法关键机制的有效性, 本实验选取当前主流大语言模型的代表版本开展对比实验, 包括 DeepSeek-r1、Claude-3.7、ChatGPT-4o 以及 Llama-4. 在确定最佳模型后, 进一步设计以下消融实验.

1) 实验 1 (SmartGen-AADL, 本文设计方案): 采用本文提出的完整技术流程, 包括架构分析、子问题分析与组件连接识别等智能体及模块, 用于评估整体性能表现.

2) 实验 2 (SmartGen-w/o SubTask, 去除子问题提取模块): 在保持其他流程不变的前提下, 移除子问题提取及组件关联分析智能体, 以评估该机制在功能细化与模块粒度控制中的作用.

3) 实验 3 (SmartGen-w/o Structure, 去除结构树构建模块): 在保留需求解析与子问题处理流程的基础上, 移除结构树构建智能体, 用于分析该模块对系统结构完整性与通信语义表达的影响.

4) 实验 4 (SmartGen-w/o RAG, 去除检索增强生成模块): 在保留需求解析、结构树构建与子问题处理流程的基础上, 移除检索增强生成模块, 用于分析该模块对系统结构完整性与通信语义表达的影响.

5) 实验 5 (LLM-direct, 直接大模型生成, 基线): 将原始需求直接输入大语言模型进行代码生成, 完全不引入中间处理流程, 用于衡量所提出方法在结构建模与语义控制方面的相对提升效果. 在该实验中, 为提高生成结果的稳定性与可控性, 提示模板中引入了 chain-of-thought 推理链和 few-shot prompting 示例引导, 通过明确要求模型按照整体概述、章节排布、组件分析、组件树构建等步骤逐层推理, 减少直接跳跃生成导致的逻辑缺陷, 同时在上下文中嵌入多个经过人工验证的 AADL 需求-架构对照示例, 使模型能够对齐生成并提升结构化输出的一致性与语法正确性. 完整提示模板包含<background><adaptive_thinking_framework><instruction><output_example><input>和<output> 等结构, 已上传至论文附录提供的 GitHub 仓库, 以确保实验的可复现性和透明度.

6 实验结果与分析

6.1 不同大语言模型实验结果对比

为了全面评估不同大语言模型在 AADL 架构生成任务中的表现, 本节从构件数量偏差度、生成代码错误行数、错误组件数、架构语义相似度以及人工评分结果等多个维度对实验结果进行了分析. 表 4 展示了不同大语言模型在各测试系统中生成构件数量的偏差情况. 从整体结果来看, 所提出的方法在大多数系统中表现出明显优势. 在 Radar 和 Ardupilot 系统中, DeepSeek-r1 模型应用本文方法后的构件数量偏差度分别为 5.26% 和 4.55%, 几乎实现了对原始架构的完全还原; 相比之下, DeepSeek-r1 模型直接生成方法的偏差度则分别高达 47.37% 和 27.27%. 这一结果充分证明了所提方法在 AADL 系统架构生成中的高还原度和有效性.

表 4 不同模型生成构件数量偏差度对比

系统	原始构件数	指标	DeepSeek-r1		Claude-3.7		ChatGPT-4o		Llama-4	
			SmartGen-AADL	LLM-direct	SmartGen-AADL	LLM-direct	SmartGen-AADL	LLM-direct	SmartGen-AADL	LLM-direct
Time_trigger	10	构件数	9	7	11	8	11	7	9	6
		偏差度 (%)	10.00	30.00	10.00	20.00	10.00	30.00	10.00	40.00
VxWorks	9	构件数	12	13	14	13	13	11	16	14
		偏差度 (%)	33.33	44.44	55.56	44.44	44.44	22.22	77.78	55.56
Radar	19	构件数	20	10	21	16	20	12	17	10
		偏差度 (%)	5.26	47.37	10.53	15.79	5.26	36.84	10.53	47.37
Car	15	构件数	17	22	17	17	18	20	19	17
		偏差度 (%)	13.33	46.67	13.33	13.33	20.00	25.00	26.67	13.33
ROSACE	38	构件数	33	16	40	42	38	35	39	24
		偏差度 (%)	13.16	57.89	5.26	5.26	5.26	7.89	2.63	41.17
Ardupilot	22	构件数	21	16	21	23	23	27	28	31
		偏差度 (%)	4.55	27.27	4.55	4.55	4.55	22.73	27.27	40.91
Pacemaker	8	构件数	9	8	9	8	10	10	8	6
		偏差度 (%)	12.50	0	12.50	0	25.00	25.00	0	25.00
Air_conditioner	12	构件数	14	9	13	11	14	11	12	9
		偏差度 (%)	16.67	25.00	8.33	8.33	16.67	8.33	0	25.00
Satellite	25	构件数	26	20	26	22	21	29	31	20
		偏差度 (%)	4.00	20.00	4.00	12.00	16.00	16.00	24.00	20.00
Solar_car	30	构件数	17	17	33	23	27	25	31	27
		偏差度 (%)	43.33	43.33	10.00	23.33	10.00	16.67	3.33	10.00
Redundancy	11	构件数	15	13	14	12	13	7	16	7
		偏差度 (%)	36.36	18.18	27.27	9.09	18.18	36.36	45.45	36.36
End_to_end	21	构件数	20	20	22	18	21	16	23	17
		偏差度 (%)	4.76	4.76	4.76	14.29	0	23.81	9.52	19.05
Periodic_task	9	构件数	9	8	9	9	9	8	13	13
		偏差度 (%)	0	11.11	0	0	0	11.11	44.44	44.44
LAMP	14	构件数	14	14	17	15	15	11	16	11
		偏差度 (%)	0	0	21.43	7.14	7.14	21.43	14.29	21.43
Control_sys	7	构件数	20	31	13	14	11	9	15	10
		偏差度 (%)	185.71	342.86	85.71	100.00	57.14	28.57	114.29	42.86
平均偏差度 (%)			25.53	47.93	18.22	18.50	15.98	22.13	27.35	32.17

此外, 不同模型在应用 SmartGen-AADL 方法后的效果差别较大, DeepSeek-r1 应用本文方法在生成架构数量偏差度上的表现最为稳定, 其生成结果在绝大多数系统中偏差度低且波动小, 显示了其在复杂系统需求和组件结

构匹配方面的优势. Claude-3.7 模型应用本文方法后的表现, 在部分系统中偏差较大, 如 VxWorks 和 Redundancy 系统的偏差度分别达到 55.56% 和 27.27%, 表明该模型在组件数量还原上存在一定波动. 在应用 SmartGen-AADL 方法后, ChatGPT-4o 模型整体表现良好, 生成的构件数量在多数系统中几乎无偏差, 且整体稳定性优于 Llama-4 模型. 相比之下, Llama-4 模型在多个系统实验中出现较大偏差, 其中在 VxWorks 系统和 Redundancy 系统中, 偏差度分别高达 77.78% 和 45.45%. 这一对比表明, 尽管两种模型均能在部分系统中实现准确的构件数量生成, 但 ChatGPT-4o 在整体表现和结果稳定性方面具有明显优势.

需要着重指出的是, Control_sys 系统在多个模型生成结果中出现偏差度异常, 其中 DeepSeek-r1 直接生成达到 342.86%, Llama-4 应用本文 SmartGen-AADL 方法后偏差度亦高达 114.29%. 尽管应用 SmartGen-AADL 方法在一定程度上降低了偏差, 但其偏差度仍然较高. 我们认为, 这一现象可能源于该系统原始构件数量较少却承载了较多功能, 且组件依赖关系复杂, 导致生成模型在捕捉组件约束和数量时更易出现严重偏差; 同时, 系统本身设计可能存在构件数量与功能分配不均的情况. 由于这一异常属于个例, 并不具有普遍性, 因此并不能否定所提出方法在整体上的有效性与稳健性.

综上, SmartGen-AADL 在整体上有效降低了大模型生成架构的偏差度, 并在复杂系统中展现出较强的稳健性. 其中, DeepSeek-r1 在应用该方法后表现最为优异, 偏差度整体最低且波动最小, 显示其在组件数量还原和复杂依赖关系建模方面的显著优势. 表 5 展示了不同模型在各测试系统中生成错误代码行数的对比情况. 整体来看, 应用 SmartGen-AADL 方法能够显著降低错误率, 而对应的直接大模型生成方法的错误率通常在 20%–40% 区间, 部分复杂系统甚至超过 50%, 表明中间处理流程在提升代码质量方面具有关键作用.

表 5　应用不同模型的错误代码行数对比 (%)

系统	DeepSeek-r1		Claude-3.7		ChatGPT-4o		Llama-4	
	SmartGen-AADL	LLM-direct	SmartGen-AADL	LLM-direct	SmartGen-AADL	LLM-direct	SmartGen-AADL	LLM-direct
Time_trigger	8.11	34.04	0	9.45	1.07	22.13	3.73	18.38
VxWorks	10.07	21.81	12.17	12.09	2.87	16.79	8.50	24.60
Radar	8.75	17.70	2.76	3.46	7.14	29.49	5.86	23.45
Car	10.53	21.39	1.14	5.20	3.72	19.89	12.01	41.67
ROSACE	4.22	21.43	2.11	6.36	5.78	13.75	8.91	10.32
Ardupilot	7.69	23.11	0	15.45	4.90	17.58	13.44	18.99
Pacemaker	2.17	11.61	1.47	2.11	5.15	14.62	4.01	18.85
Air_conditioner	11.06	35.59	1.14	5.20	3.72	19.89	24.45	26.52
Satellite	7.55	20.99	2.45	9.28	6.10	1.73	1.65	19.85
Solar_car	7.89	45.11	3.23	11.88	3.89	18.18	16.02	26.34
Redundancy	6.56	34.55	0.87	9.63	7.98	16.39	12.89	20.31
End_to_end	2.77	21.67	1.50	4.36	5.33	28.99	10.37	36.31
Periodic_task	3.10	16.53	2.16	5.24	3.01	22.76	9.09	27.78
LAMP	2.57	26.84	1.33	6.28	7.06	23.87	14.99	25.83
Control_sys	12.82	50.67	2.67	6.25	7.69	27.63	12.74	31.48
平均值	7.06	26.87	2.33	7.48	5.03	19.58	10.58	24.71

通过详细分析实验结果发现, Claude-3.7 在 SmartGen-AADL 方法下表现最为突出, 平均错误率仅为 2.33%, 并在 Time_trigger 与 Ardupilot 系统中实现零错误, 显示其在结构清晰、任务规模适中的场景中具有极高的准确性与稳定性. 相比之下, DeepSeek-r1、ChatGPT-4o 与 Llama-4 的平均错误率分别为 7.06%、5.03% 和 9.02%, 整体稳定性存在差异, 其中 ChatGPT-4o 与 DeepSeek-r1 在应用 SmartGen-AADL 前后的性能改善更为明显, Llama-4 的平均错误率为 9.02%, 即便结合 SmartGen-AADL 仍显著高于其他模型, 在 Air_conditioner 与 Solar_car 等复杂系统中错误率超过 15%, 表现出明显的不稳定性. 值得注意的是, Control_sys 系统在多个模型生成结果中依旧出现较高错误率, 例如 DeepSeek-r1 直接生成达到 50.67%, 这一异常与其构件数量少、功能承载重及依赖关系复杂密切相关.

SmartGen-AADL 在错误组件数方面也同样始终优于直接大模型生成方法. 由表 6 可见, 无论采用哪种大语言模型, 应用本文所提出的方法均能显著降低错误率. 例如, 在 Time_trigger 系统中, DeepSeek-r1 的错误率由 77.78% 降至 21.05%; 在 Ardupilot 系统中, 应用本文方法后, Claude-3.7 实现了零错误; 在 VxWorks 系统中, 应用本文方法后, ChatGPT-4o 的错误率由 69.23% 降至 11.76%; 而在 Car 系统中, Llama-4 的错误率则由 73.68% 降至 3.75%. 这些结果表明, 直接生成方法常因缺乏结构约束而导致组件遗漏或冗余, 而本文提出的方法通过子任务分解、结构树构建和语义关联识别等机制, 有效提升了生成结果的完整性和一致性.

表 6　应用不同模型的错误组件数对比 (%)

系统	DeepSeek-r1		Claude-3.7		ChatGPT-4o		Llama-4	
	SmartGen-AADL	LLM-direct	SmartGen-AADL	LLM-direct	SmartGen-AADL	LLM-direct	SmartGen-AADL	LLM-direct
Time_trigger	21.05	77.78	0	20.00	8.33	54.55	23.53	66.67
VxWorks	44.44	54.17	36.84	38.89	11.76	69.23	31.52	75.00
Radar	31.43	66.67	8.00	11.11	19.56	20.51	16.13	23.08
Car	41.67	83.33	14.29	15.79	30.00	35.71	3.75	73.68
ROSACE	16.22	23.53	15.56	35.71	13.79	21.42	31.11	13.33
Ardupilot	23.81	60.87	0	11.11	25.00	71.43	26.92	46.15
Pacemaker	17.02	22.22	5.26	7.69	20.00	45.45	10.63	40.00
Air_conditioner	30.00	70.00	5.56	18.18	17.39	41.67	27.59	80.00
Satellite	35.00	58.82	13.33	14.29	21.88	46.67	43.48	23.94
Solar_car	40.00	66.67	10.17	27.03	24.00	50.00	41.18	66.67
Redundancy	18.92	70.59	6.67	46.67	40.00	28.57	52.17	42.86
End_to_end	6.25	31.58	0.97	20.00	21.43	75.00	40.74	69.23
Periodic_task	12.50	72.73	8.75	15.38	7.14	25.00	29.41	64.29
LAMP	5.88	42.31	8.75	28.00	32.14	81.82	3.75	54.55
Control_sys	30.00	88.46	2.67	6.30	7.69	27.63	12.74	31.48
平均值	24.95	59.32	9.12	21.08	20.01	46.31	26.31	51.40

在不同模型的表现方面, 应用本文方法后, Claude-3.7 整体效果最优, 错误率始终维持在较低水平, 并在 Time_trigger 和 Ardupilot 等系统中实现了完全正确的组件生成, 表现出较强的适应性与鲁棒性. 然而, 在 Redundancy 等涉及复杂冗余机制的系统中, 其性能仍出现一定程度下降. DeepSeek-r1 在应用本文方法后结果波动较大, 在 ROSACE 和 Pacemaker 等系统中能够保持较低错误率, 但在 VxWorks 和 Car 等规模较大或架构复杂的系统中表现不够稳定, 说明其在高度异构场景下的一致性仍需提升. ChatGPT-4o 在应用本文方法后在部分任务中表现突出, 例如在 VxWorks、Pacemaker 和 Control_sys 等系统中均取得了较低错误率, 但在 Car 和 Redundancy 等系统中仍出现显著偏高的情况, 显示其跨系统性能的稳定性不足. 相比之下, Llama-4 即便在应用本文方法后, 整体表现依然不及其他模型, 平均错误率普遍偏高, 尤其在 End_to_end、Solar_car 和 Redundancy 等系统中效果较差. 不过在 Car 和 LAMP 等个别系统中, 其生成质量优于其他模型, 说明在特定任务下仍具备一定潜力, 但整体缺乏普适性.

在 SmartGen-AADL + Claude-3.7 组合生成的 Solar_car 系统模型中, Batteries 模块出现了如图 10 所示的代码片段.

```
data Energy_Data
properties
    Data_Model::Data_Representation => Struct;
    Source_Data_Size => 64 Bytes;
end Energy_Data;
```

图 10　SmartGen-AADL Batteries 模块代码片段

在编译分析过程中, 工具报告 Data_Representation 和 Source_Data_Size 无法被解析, 提示这些属性未指向任何有效定义或不可访问的标识符, 并最终导致模型无法被正确分析. 其根本原因在于, AADL 标准库中并不存在这两个属性项, 也未通过 with 子句从外部包中导入, 因此编译器无法在当前作用域中识别, 产生语义解析错误. 若需表达类似语义, 应在自定义包中显式声明并导入相应属性, 或使用 AADL 标准中已定义的等价属性 (如以 Data_

Size 替代 Source_Data_Size).

从生成过程来看, 该错误与原始需求描述的抽象程度及检索到的知识内容均有关系. 在 Solar_car 系统的需求文本中, Batteries 模块被描述为: “SolarCells 与 Batteries 子系统作为电源相关设备的代表, 虽然在本架构中未直接嵌入处理器与内存组件, 但它们均连接至 CAN 总线, 可实现能源管理数据的实时采集与上报, 支持系统能耗监测与自适应能量调配策略的实施.” 该描述仅说明了模块的功能定位与通信关系, 缺乏内部数据结构、属性约束及资源使用等细节. 面对这种信息不足的情况, 模型在生成阶段往往会基于上下文进行语义扩展, 将抽象的“能源数据”补充为具有属性定义的数据结构. 当扩展涉及属性层面时, 由于模型对 AADL 规范中属性体系的理解有限, 容易生成形式上合理但规范中并不存在的属性项, 从而产生语义层错误.

同时, 该错误还与检索增强生成 (RAG) 阶段的知识内容有关. 检索到的语料中提到, 用户可以在属性中反映数据建模信息, 如存储需求、数据单位、数组维度或类型层级等. 这段文本仅起到说明性作用, 并未明确哪些属性属于 AADL 规范内的有效定义. 模型在理解时误将此类示例性描述泛化为规范性定义, 进而将 Data_Representation 与 Source_Data_Size 视作可直接调用的属性. 由于检索结果中缺乏包含 Data_Size 等正确示例的语料, 模型在属性生成阶段出现了偏差, 从而导致该错误的产生.

与前一案例中属性定义错误不同, 另一类典型问题出现在系统连接关系的生成阶段. 以基于 DeepSeek-r1 的 SmartGen-AADL 生成的 Satellite 系统为例, 其中 Satellite_On_Orbit_Management 模块生成了如图 11 所示的代码片段.

```
system implementation Satellite_On_Orbit_Management_System.impl
   subcomponents
       Embedded_Processor: processor Embedded_Processor.impl;
       System_Memory: memory System_Memory.impl;
       AOCS_Bus: bus AOCS_Bus.impl;
       Platform_Bus: bus Platform_Bus.impl;
       Payload_Bus: bus Payload_Bus.impl;
       AOCS_Device: device AOCS_Device.impl;
       Platform_Device: device Platform_Device.impl;
       Payload_Device: device Payload_Device.impl;
       OBSW_Process: process OBSW_Process.impl;
   connections
       command_conn: port command_in -> OBSW_Process.telecommand_in;
       telemetry_conn: OBSW_Process.telemetry_out -> port telemetry_out;
end Satellite_On_Orbit_Management_System.impl;

bus AOCS_Bus
   features
       bus_in: in data port;
       bus_out: out data port;
end AOCS_Bus;
```

图 11 Satellite 系统代码片段

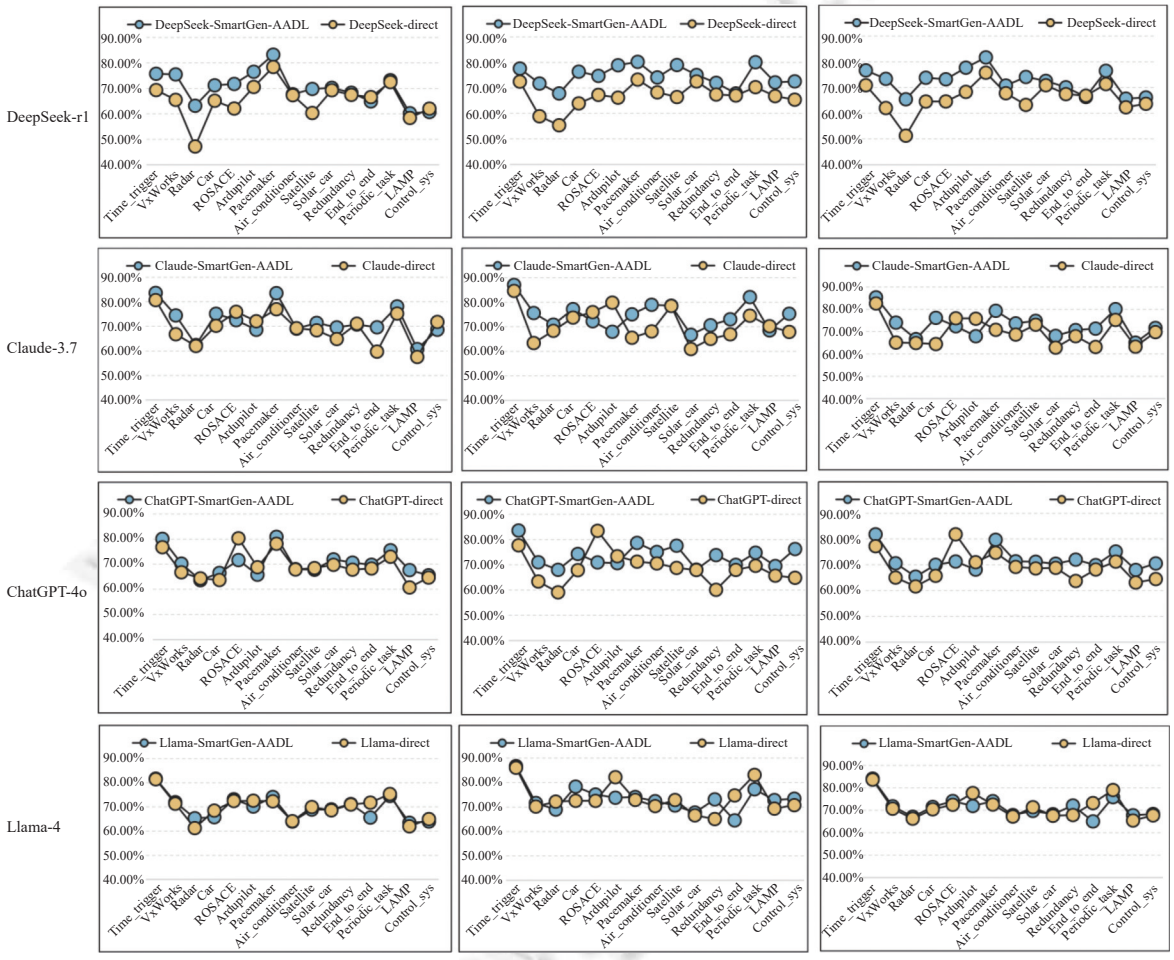
在系统解析过程中, 编译器报告了多处语法与语义错误. 首先, 在 connections 段落中, 语句 telemetry_conn: OBSW_Process.telemetry_out -> port telemetry_out; 的格式不符合 AADL 规范, 导致解析时出现“unexpected token”错误. 根据标准语法, 端口应在组件特征区中单独声明, 而非直接在连接表达式中引用. 其次, 在 Bus 段落中, 模型错误地为 AOCS_Bus 组件添加了数据端口 bus_in 与 bus_out, 而总线类型组件并不允许直接声明数据端口. 按照 AADL 规范, 总线是用于通信的介质, 其交互应通过其他组件的 requires bus access 特征来实现, 例如 SignalWire: requires bus access Wire.Gauge12, 而不是在 Bus 组件内部定义数据流端口.

该错误类型反映了模型在生成跨组件结构关系时的语义理解偏差, 特别是在连接 (connection)、绑定 (bind) 和数据流 (port) 等涉及多层交互的结构中更容易出现问题. 其根源在于通用大模型对 AADL 语言的层次关系和语法约束掌握不足, 未能准确区分 System、Process、Device、Bus、Processor、Memory 等不同组件在连接方式上的语义边界. 例如, bind 用于定义软件与硬件的绑定关系, 必须在系统层次声明, 而非局部组件中实现. 模型在生成时往往依据自然语言表述逻辑直接推断组件间的关联, 从而忽略了 AADL 的严格层级约束与类型限定. 此外, 生成链中所用的 AADL 校验工具仅提供基础语法级提示, 缺乏针对语义层面的反馈. 例如, 错误信息虽能指出“unexpected token”或“cannot have port spec as a feature”, 但未说明合法的连接格式或正确的特征声明方式, 导致模型虽能发现

错误, 却难以依此调整生成策略, 最终使复杂系统结构中的连接错误得以保留.

综上所述, SmartGen-AADL 在各测试系统中整体表现出显著的错误率降低与结果稳定性提升. 其中, DeepSeek-r1 在编译通过率的稳定性方面最为突出, Claude-3.7 在平均准确性上表现最佳, 而 ChatGPT-4o 与 Llama-4 在处理部分复杂系统时仍显示出一定局限性. 上述结果表明, 所提出的方法能够有效提升大模型生成代码的质量, 但在依赖关系高度密集的复杂系统中仍存在进一步优化的空间.

图 12 展示了不同模型在各测试系统上架构语义相似度的对比结果, 包括 R_{BERT}、P_{BERT} 与 F_{BERT} 这 3 种指标. 整体来看, 应用本文所提出的方法后, 各模型在绝大多数系统中的语义相似度均得到显著提升, 相较于直接大模型生成方法, 平均提升幅度在 5%–10% 之间, 表明中间处理流程对于保持系统架构语义完整性与精确表达具有关键作用. 直接生成方法在部分复杂系统中表现不稳定, 例如在 Radar、VxWorks 和 End_to_end 系统中语义相似度出现明显下降, 反映了缺少子问题分解与结构建模辅助时模型容易生成语义冗余或不一致的架构组件.



(a) R_{BERT} (b) F_{BERT} (c) P_{BERT}

图 12 应用不同模型架构的语义相似度对比

从模型表现角度分析, Claude-3.7 模型应用本文方法后的语义相似度整体较高, 但波动明显. 在 R_{BERT} 指标下, 虽然 Time_trigger、Pacemaker 和 Periodic_task 等结构清晰、任务规模适中的系统语义匹配率超过 85%, 显示其在中等复杂度场景中能够较准确地捕获系统组件关系, 但在 VxWorks 和 Solar_car 系统中, 语义相似度分别下降至 70.23% 和 66.76%, 表明在复杂依赖或组件数量较多的系统中存在明显波动. P_{BERT} 和 F_{BERT} 指标下类似情况亦

存在, 例如 P_{BERT} 指标下 Solar_car 语义匹配率仅为 66.76%, F_{BERT} 指标下 End_to_end 系统下降至 66.34%, 说明 Claude-3.7 在处理高度复杂或组件密集型系统时表现不够稳定.

相比之下, DeepSeek-r1 应用本文方法后表现更加均衡和稳健. 无论是 R_{BERT}、P_{BERT} 还是 F_{BERT} 指标, 其在绝大多数系统的语义相似度均保持在 70% 以上, 在 Radar、Ardupilot 和 End_to_end 等任务复杂系统中仍能维持较高匹配率, 如 Radar 系统在 R_{BERT} 指标下达到 63.20%, Ardupilot 在 P_{BERT} 指标下达到 78.90%, End_to_end 在 F_{BERT} 指标下保持 66.34%. 这些数据表明, DeepSeek-r1 在面对复杂系统组件关系和任务调度时, 能够有效保持语义完整性与生成稳定性, 其性能波动远低于 Claude-3.7. 进一步对比直接生成方法, 可以看到, Claude-3.7 直接生成时的波动更为显著. 例如 R_{BERT} 指标下 Radar 系统仅为 62.02%, P_{BERT} 指标下 VxWorks 下降至 58.91%, F_{BERT} 指标下 End_to_end 仅为 63.13%, 与应用本文方法后的表现相比, 提升幅度分别达到 0.38、12.82 和 3.21 个百分点, 显示中间结构分析和子问题分解对稳定生成高质量架构具有显著作用.

综合来看, Claude-3.7 在语义相似度上尽管在部分系统中能够达到较高水平, 但其性能波动较大, 尤其在复杂系统中存在不稳定现象; 而 DeepSeek-r1 应用本文方法后表现最为稳定, 能够在不同复杂度系统中保持均衡且可靠的语义匹配效果, 凸显其在复杂系统架构生成任务中的优势.

表 7 展示了不同模型在各测试系统中生成架构的人工评分结果. 整体来看, 应用本文提出的 SmartGen-AADL 方法后, 所有模型的评分均显著高于直接生成方式, 普遍提升约 0.3–0.7 分. 这表明中间处理流程在保持架构合理性与一致性方面发挥了重要作用. 相比之下, 直接生成方法在复杂系统如 Car、End_to_end 与 Control_sys 中得分明显偏低, 反映出在缺少结构约束时模型容易产生结构不完整或逻辑错误的组件.

表 7　人工评分结果与分析结果统计

系统	DeepSeek-r1		Claude-3.7		ChatGPT-4o		Llama-4	
	SmartGen-AADL	LLM-direct	SmartGen-AADL	LLM-direct	SmartGen-AADL	LLM-direct	SmartGen-AADL	LLM-direct
Time_trigger	4.31	3.96	4.89	4.25	4.71	4.10	4.38	4.02
VxWorks	4.53	3.94	4.56	4.39	4.66	4.21	4.47	3.99
Radar	4.1	3.76	4.69	4.52	4.53	4.00	4.32	3.87
Car	4.55	3.89	4.81	4.76	4.70	4.33	4.10	3.45
ROSACE	4.46	3.75	4.59	4.32	4.33	4.08	4.11	3.88
Ardupilot	4.67	4.09	4.91	4.11	4.51	4.05	4.25	3.92
Pacemaker	4.53	3.97	4.78	4.51	4.34	4.46	3.88	3.51
Air_conditioner	4.18	3.43	4.43	4.30	4.31	4.19	3.93	3.75
Satellite	4.23	4.01	4.64	4.18	4.12	4.47	4.57	4.03
Solar_car	4.39	3.48	4.62	4.37	4.62	4.26	4.34	3.91
Redundancy	4.68	3.65	4.82	4.61	4.75	3.99	3.90	3.84
End_to_end	4.71	4.05	4.72	4.31	4.27	4.01	4.65	4.09
Periodic_task	4.46	3.7	4.53	4.13	4.50	3.92	3.64	3.61
LAMP	4.62	3.66	4.67	4.44	4.18	4.06	3.88	3.42
Control_sys	3.95	3.47	4.62	4.20	4.19	3.65	3.86	3.44
平均值	4.42	3.79	4.69	4.36	4.45	4.12	4.15	3.78

从模型对比的角度来看, Claude-3.7 在应用 SmartGen-AADL 后表现最为突出, 平均评分达到 4.62 分, 显著高于其他模型. 其在 Time_trigger、Ardupilot、Redundancy 等多个系统中的评分接近 5 分, 显示其在架构清晰、任务规模适中的场景下能够生成高质量且稳定的系统模型. DeepSeek-r1 的平均得分为 4.42 分, 整体表现稳定, 在大多数系统中保持了良好的输出一致性. ChatGPT-4o 的平均得分为 4.45 分, 整体优于 Llama-4, 但在 Control_sys 与 LAMP 等复杂系统中的得分仍低于 4.2 分, 表明在处理复杂依赖关系时存在一定不足. Llama-4 的平均得分最低, 仅为 4.15 分, 在部分系统中仍未能突破 4.0 分, 显示其在高复杂度场景下的可读性与合理性相对较弱.

综上所述, Claude-3.7 在人工评分中表现最佳, 既具备较高的准确性, 又展现了较好的稳定性; DeepSeek-r1 和 ChatGPT-4o 的整体表现居中, 前者在不同系统上更为稳定, 后者在部分场景中略显波动; Llama-4 的结果则相对不

理想, 在复杂系统中仍存在明显的优化空间.

通过构件数量匹配、代码生成错误率、语义匹配效果及人工评分这 4 个维度的实验对比可以看出, DeepSeek-r1 在组件数量还原、语义稳定性和编译通过率等方面均表现最为稳健, 能够在不同复杂度的系统中保持可靠的输出, 是整体稳健性与通用性最优的模型; 而 Claude-3.7 则在平均准确性和人工评分中占据优势, 展现了更高的精度和代码可读性, 但在复杂系统中稳定性略显不足, 因此可以认为 DeepSeek-r1 是综合最优的稳健模型, 而 Claude-3.7 则在生成质量与准确性上更具优势.

6.2 基于 DeepSeek-r1 的 SmartGen 消融实验

为了系统评估各实验方案在生成 AADL 代码时的结构规模还原能力与质量表现, 本节以 DeepSeek-r1 模型作为主要智能体, 基于前文所述的 4 组实验设计分别生成 AADL 架构代码. 通过消融关键模块 (子问题提取、结构树构建及 RAG 知识检索) 与端到端生成方案的对比, 分析各实验组在构件数量偏差度、错误行占比、错误组件占比及人工评分等指标上的表现差异, 从而验证完整方法在控制系统规模、保证结构一致性及提升工程可用性方面的有效性.

6.2.1 基于 DeepSeek-r1 的 SmartGen 消融实验构件数量偏差度评估结果

为量化各实验方案在结构规模还原上的能力, 对比分析了不同实验组生成架构中构件数量与原始 AADL 设计之间的偏差情况. 如表 8 所示的实验结果显示, 本文设计的完整方法 (SmartGen-AADL) 在该指标上表现稳定, 15 个系统的平均偏差度为 25.53%, 说明所提方法能够较好地控制生成组件数, 保持与参考架构在规模层面的接近性. 然而, 在移除组件结构树构建智能体的 SmartGen-w/o Structure 实验设置中, 偏差度显著上升, 反映出结构树机制在系统规模约束方面的核心作用.

表 8 基于 DeepSeek-r1 的 SmartGen 消融实验构件数量偏差度对比

系统	原始构件数	指标	SmartGen-AADL	SmartGen-w/o SubTask	SmartGen-w/o Structure	SmartGen-w/o RAG	LLM-direct
Time_trigger	10	构件数	9	9	9	8	7
		偏差度 (%)	10.00	10.00	10.00	20.00	30.00
VxWorks	9	构件数	12	12	14	15	13
		偏差度 (%)	33.33	33.33	55.56	33.33	44.44
Radar	19	构件数	20	20	16	17	10
		偏差度 (%)	5.26	5.26	15.79	10.53	47.37
Car	15	构件数	17	17	25	26	22
		偏差度 (%)	13.33	13.33	66.67	73.33	46.67
ROSACE	38	构件数	33	33	28	35	16
		偏差度 (%)	13.16	13.16	26.32	7.89	57.89
Ardupilot	22	构件数	21	21	18	19	16
		偏差度 (%)	4.55	4.55	18.18	13.64	27.27
Pacemaker	8	构件数	9	9	9	10	8
		偏差度 (%)	12.50	12.50	12.50	25.00	0
Air_conditioner	12	构件数	14	14	9	14	9
		偏差度 (%)	16.67	16.67	25.00	16.67	25.00
Satellite	25	构件数	26	26	20	28	20
		偏差度 (%)	4.0	4.0	20.0	12.00	20.0
Solar_car	30	构件数	17	17	16	19	17
		偏差度 (%)	43.33	43.33	46.67	36.67	43.33
Redundancy	11	构件数	15	15	9	15	13
		偏差度 (%)	36.36	36.36	18.18	36.36	18.18
End_to_end	21	构件数	20	20	10	22	20
		偏差度 (%)	4.76	4.76	52.38	4.76	4.76

表 8 基于 DeepSeek-r1 的 SmartGen 消融实验构件数量偏差度对比 (续)

系统	原始构件数	指标	SmartGen-AADL	SmartGen-w/o SubTask	SmartGen-w/o Structure	SmartGen-w/o RAG	LLM-direct
Periodic_task	9	构件数	9	9	7	9	8
		偏差度 (%)	0	0	22.22	0	11.11
LAMP	14	构件数	14	14	14	15	14
		偏差度 (%)	0	0	0	7.14	0
Control_sys	7	构件数	20	20	19	23	31
		偏差度 (%)	185.71	185.71	171.43	228.57	342.86
平均偏差度 (%)			25.53	25.53	37.39	35.06	47.93

结构复杂度较高的系统对连接关系识别尤为敏感. 在 ROSACE、Radar 以及 Car 等多层级、通信耦合强的系统中, SmartGen-w/o Structure 的偏差度分别达到 26.32%、15.79% 和 66.67%, 远高于 SmartGen-AADL 在相应系统中的表现 (分别为 13.16%、5.26% 和 13.33%). 在这些系统中, 原始 AADL 架构通常包含多个子模块和接口链路, 缺乏结构树的引导时, 大模型倾向于将相邻功能块误解为独立组件进行重复生成, 或因缺乏连接信息的约束将逻辑相关的结构错误拆分, 最终导致组件数量膨胀或划分失真. 这一现象表明, 仅依靠自然语言中的隐式语义信号难以支撑复杂系统的粒度划分与拓扑复现, 连接关系的显式建模是保持结构一致性的前提. 在部分系统中, 构件数量偏差度相对较小, 尤其是在 Pacemaker 与 Time_trigger 等功能边界清晰的系统中, SmartGen-AADL 与 SmartGen-w/o Structure 的偏差度分别仅为 12.50% 和 10.00%. 这说明当自然语言需求本身具有明确的模块指示语义时, 结构树的缺失对生成结果的影响相对有限, 模型在无额外结构指导的情况下也能较为准确地完成划分. 但即便如此, 在其他中等复杂度系统中, 如 Redundancy 与 Air_conditioner, SmartGen-w/o Structure 依然出现了不同程度的偏差, 进一步提示该机制在维护连接合理性、抑制无效组件生成方面具有普适价值.

值得注意的是, 在极端场景 Control_sys 系统中, SmartGen-w/o Structure 虽然移除了系统结构树, 但仍保持在 19 个组件, 而直接由 LLM 生成的实验, 即 LLM-direct 的构件数则激增至 31 个, 偏差度达到 3.429, 呈现出无连接指导下系统规模失控的典型特征. 这一结果表明, 即便在保留需求解析与子问题提取的基础上, 仅引入结构树构建这一环节, 亦能显著压制组件数量的异常扩张趋势, 从而确保系统架构具备可部署的结构约束性. 同样, LLM-direct 在多个复杂系统中表现出显著偏差. 例如, 在 Control_sys 系统中, 生成系统构件数达 31 个, 远超原始设计的 7 个, 偏差度高达 3.428 6, 说明在缺乏结构感知机制的条件下, LLM 容易产生组件重复定义或对抽象描述过度拆分的问题. 类似现象也出现在 ROSACE、Radar 等系统中, 偏差度分别达到 0.578 9 与 0.473 7, 显示该策略在高复杂度系统上的结构建模能力存在明显不足.

在移除 RAG 模块的 SmartGen-w/o RAG 实验中, 整体构件数量偏差有所增加, 平均偏差度达到 0.479 3, 高于完整方法的 0.255 3. 具体来看, 该设置在 Radar、Car、ROSACE 等多层级、耦合度高的系统中偏差度分别达到 10.53%、73.33% 和 35.06%, 均高于 SmartGen-AADL 的表现. 这表明 RAG 模块在提供外部知识检索与上下文补充方面对复杂系统的生成具有显著作用, 其缺失容易导致模型在面对抽象需求或信息不充分的场景时, 产生组件遗漏、重复或划分不当, 从而加大构件数量偏差. 相比之下, 结构树仍能在一定程度上维持系统的基本拓扑, 但缺乏 RAG 的语义增强, 生成结果的精确性与鲁棒性仍受到明显影响.

此外, 还观察到部分系统在所有实验组中普遍存在较高偏差, 如 Redundancy 与 Control_sys. 此类系统原始组件数量较少, 结构粒度较粗, 自然语言表达中存在较多未明确界定的功能逻辑或通信路径, 极易在需求解析阶段被过度拆解. 大模型在应对这类输入时, 通常倾向于语义泛化或自动补全“潜在必要组件”, 例如冗余传感器、状态监测器或中间控制逻辑, 从而造成生成结果中组件数量明显高于设计预期. 另一方面, 构件数量偏差度较低的任务往往具备更强的结构规范性和功能边界清晰度. 例如, 在 Pacemaker 与 Time_trigger 这类功能明确、接口层次单一的系统中, 生成模型在所有实验配置下的表现均较为稳定, 构件偏差度维持在较低水平. 相较之下, Solar_car 与 Control_sys 等系统中, 由于原始需求文本冗长、功能描述抽象或包含大量泛化术语, 模型更容易在组件边界识别上产生混淆, 进而导致误分、漏分或非必要组件的冗余生成.

总之, 组件连接关系识别机制, 即结构树构建过程, 不仅在结构复杂、层级深的系统中发挥着关键的结构约束作用, 也在中低复杂度场景中有效增强了系统生成过程的鲁棒性. 在缺失该模块的条件下, 生成模型在组织能力与结构一致性方面显著下降, 特别是在需求表达抽象程度较高或语义线索不充分的场景中, 构件数量偏差问题尤为突出, 严重影响了系统架构的还原精度与可部署性.

6.2.2 基于 DeepSeek-r1 的 SmartGen 消融实验错误率指标对比分析

为了进一步评估各方案在生成代码质量与组件结构可靠性方面的差异, 本节对错误行占比与错误组件占比两类指标在不同处理机制下的表现进行了系统分析. 实验结果分别如表 9 及表 10 所示.

表 9 基于 DeepSeek-r1 的 SmartGen 消融实验错误行占比情况

系统	SmartGen-AADL			SmartGen-w/o SubTask			SmartGen-w/o Structure			SmartGen-w/o RAG			LLM-direct		
	错误	总数	L_{error} (%)	错误	总数	L_{error} (%)	错误	总数	L_{error} (%)	错误	总数	L_{error} (%)	错误	总数	L_{error} (%)
Time_trigger	18	222	8.11	12	142	8.45	15	156	9.62	17	172	8.14	32	94	34.04
VxWorks	29	288	10.07	33	269	12.27	31	270	11.48	30	270	11.11	41	188	21.81
Radar	30	343	8.75	40	307	13.03	53	328	16.16	33	310	10.65	20	113	17.70
Car	32	304	10.53	42	298	14.09	48	322	14.91	51	350	14.57	43	201	21.39
ROSACE	19	450	4.22	5	375	1.33	37	291	12.71	22	431	5.10	42	196	21.43
Ardupilot	24	312	7.69	23	309	7.44	31	288	10.76	23	336	6.85	49	212	23.11
Pacemaker	9	414	2.17	29	231	12.55	29	223	13.00	17	409	4.16	13	112	11.61
Air_conditioner	22	199	11.06	25	212	11.79	21	173	12.14	25	219	11.41	42	118	35.59
Satellite	40	530	7.55	100	505	19.80	28	326	8.59	39	511	7.63	38	181	20.99
Solar_car	27	342	7.89	34	245	13.88	25	245	10.20	29	307	9.45	83	184	45.11
Redundancy	25	381	6.56	59	523	11.28	24	316	7.59	28	396	7.01	57	165	34.55
End_to_end	9	325	2.77	23	285	8.07	23	285	8.07	11	312	3.53	57	263	21.67
Periodic_task	7	226	3.10	8	166	4.82	15	152	9.87	8	223	3.59	20	121	16.53
LAMP	10	389	2.57	8	186	4.30	37	230	16.09	9	358	2.51	62	231	26.84
Control_sys	10	78	12.82	57	380	15.00	71	372	19.09	17	125	13.60	114	225	50.67
平均错误率 (%)		7.06			10.54			12.02			7.95			26.87	

表 10 基于 DeepSeek-r1 的 SmartGen 消融实验错误组件占比情况

系统	SmartGen-AADL			SmartGen-w/o SubTask			SmartGen-w/o Structure			SmartGen-w/o RAG			LLM-direct		
	错误	总数	C_{error} (%)	错误	总数	C_{error} (%)	错误	总数	C_{error} (%)	错误	总数	C_{error} (%)	错误	总数	C_{error} (%)
Time_trigger	4	19	21.05	6	17	35.29	6	16	37.50	5	20	25.00	7	9	77.78
VxWorks	12	27	44.44	10	18	55.56	15	26	57.69	10	21	47.92	13	24	54.17
Radar	11	35	31.43	19	33	57.58	20	36	55.56	15	41	36.59	8	12	66.67
Car	10	24	41.67	8	19	42.11	11	18	61.11	12	24	50.00	15	18	83.33
ROSACE	6	37	16.22	3	36	8.33	11	28	39.29	9	39	23.08	4	17	23.53
Ardupilot	10	42	23.81	15	36	41.67	17	33	51.52	12	40	30.00	14	23	60.87
Pacemaker	8	47	17.02	8	23	34.78	8	22	36.36	11	49	22.49	2	9	22.22
Air_conditioner	6	20	30.00	6	22	27.27	6	16	37.50	5	19	26.32	7	10	70.00
Satellite	21	60	35.00	33	46	71.74	8	20	40.00	28	57	49.12	10	17	58.82
Solar_car	10	25	40.00	12	21	57.14	9	21	42.86	10	26	38.46	12	18	66.67
Redundancy	7	37	18.92	15	45	33.33	6	27	22.22	8	33	24.24	12	17	70.59
End_to_end	2	32	6.25	9	28	32.14	9	28	32.14	6	35	17.14	12	38	31.58
Periodic_task	3	24	12.50	4	12	33.33	5	11	45.45	4	25	16.00	8	11	72.73
LAMP	3	51	5.88	4	38	10.53	11	30	36.67	4	49	8.16	11	26	42.31
Control_sys	3	10	30.00	18	32	56.25	21	35	60.00	5	13	38.46	23	26	88.46
平均错误率 (%)		24.95			39.80			43.72			30.20			59.32	

整体结果表明, SmartGen-AADL 在大多数任务中均保持低位且集中分布, 而缺乏关键机制的变体以及直接生成方案则呈现出更高且波动显著的错误密度. 在不同系统任务中, SmartGen-AADL 所生成的错误行占比普遍控制在较低水平, 大多数任务中的错误行占比维持在 10% 以下, 错误组件占比则大多集中在 30% 左右甚至更低, 表现出良好的可控性与稳定性. 相较之下, LLM-direct 的错误率波动范围较大, 多个任务中出现显著偏高的错误密集区域, 其中 Solar_car、Control_sys、Time_trigger 等系统的错误组件占比超过一半, 生成代码在可部署性与可维护性方面显著受限. 在去除 RAG 模块的 SmartGen-w/o RAG 实验中, 整体错误行占比与错误组件占比均出现不同程度的上升, 平均错误行占比和错误组件占比明显高于完整方法, 但仍优于直接端到端生成方案. 具体来看, 该设置在 Radar、Car、ROSACE 等高复杂度系统中的错误行占比分别为 10.65%、14.57% 和 5.10%, 组件错误占比分别达到 36.59%、50.00% 和 23.08%, 均高于 SmartGen-AADL 对应指标. 分析可知, RAG 模块通过提供外部知识检索与上下文补充, 有效辅助模型理解复杂需求与架构约束, 其缺失会导致模型在处理抽象描述、接口绑定及组件依赖时出现遗漏或划分不当, 从而增加错误密度. 该结果表明, RAG 模块在降低代码生成错误率、提高组件结构可靠性方面发挥了关键作用, 是保证模型生成高质量 AADL 代码的重要机制.

值得注意的是, 即便在采用完整流程仍有个别系统出现相对偏高的错误率. 例如 Air_conditioner 同时包含复杂的调度链与能耗约束, 且整体代码行数并不庞大, 因而只需少量语法或属性声明失误便会显著抬高错误比例. 尽管如此, 完整方法在绝大多数任务中的错误行占比仍稳居低个位数区间, 错误组件占比亦多集中在 30% 以下, 与直接端到端生成方案相比, 保持了明显的质量优势. 该差异主要源于 LLM 对 AADL 语法及架构建模知识的覆盖不足: 模型在处理端口、属性及层级引用等强结构化元素时易产生遗漏或冲突. 本文知识库构建能够使模型在生成过程中能够实时检索并注入 AADL 领域知识; 该能力与“子问题提取”和“结构树构建”机制协同, 为模型同时提供任务粒度约束与拓扑先验, 从而弥补大模型在 AADL 语法和架构要素覆盖不足方面的短板.

此外, 本文发现, 在需求结构较清晰的系统中, 如 Time_trigger 与 Pacemaker, 尽管不同实验配置间仍存在一定差异, 但整体错误率波动相对较小. 这一结果表明, 当输入需求本身具备良好的语义分区特性时, 语言模型较容易从中提取结构线索, 即使缺乏部分辅助机制, 其结构生成质量亦能保持在合理水平. 然而, 即便在此类场景中, 完整方法仍表现出更低的错误率, 体现出辅助机制在增强鲁棒性与泛化能力方面的稳定作用. 对于结构复杂度较高的系统, 错误控制能力则更依赖模型结构感知能力与需求分解机制的配合. 以 Radar 为例, 完整方法生成的错误行占比为 8.75%, 错误组件占比为 31.43%; 而在去除结构树构建的 SmartGen-w/o Structure 中, 两项指标分别上升至 16.16% 与 55.56%. 分析生成内容可发现, 该差异主要源于模型在处理连接路径时出现未定义引用、端口绑定不完整等问题, 说明在高耦合任务中, 结构连接机制对于约束组件层级与构建语义路径具有不可替代的重要作用. 此外, 部分系统本身结构粒度较粗、组件数量较少 (如 Control_sys、Redundancy), 在自然语言表达冗余或语义提示不足的情况下, 生成模型容易出现误分、合并失当等现象, 从而推高错误比率. 尽管如此, SmartGen-AADL 的完整方法通过结构解析与语义细化, 虽然仍存在一定错误, 但整体结构可读性与可用性显著改善, 证明即使在语义线索极为稀疏的情况下, 引入结构组织与分层生成策略依然能够发挥重要作用.

本文对报错日志进行了分析, 发现 SmartGen-AADL 的错误行分布呈“长尾”特征——绝大多数模块零散出现单行或少量编译错误, 集中在接口关键字拼写、调度参数单位等细节; LLM-direct 则常见大片段无法解析, 尤其在定义子程序或 thread implementation 时易出现成批未闭合或多重声明类错误.

通过对实验结果分析发现, 在实际系统生成过程中, 子问题提取和组件间关系分析智能体在确保代码正确性与组件语义一致性方面发挥了关键作用. 该机制能够有效将需求中的复杂语句拆解为若干语义明确、作用范围清晰的功能子目标, 并在代码生成阶段逐一绑定到对应的 AADL 实体上, 从而实现细粒度的结构控制与属性精确分配. 以 LAMP 在线分析系统为例, 需求中明确提出任务处理核心模块 task_processing_core 需要以固定周期运行并满足一系列实时调度要求, 在引入子问题提取机制的完整方法 (SmartGen-AADL) 中, 模型能够准确识别这些调度语义, 并将其绑定到系统的各个线程级别的组件上. 例如, 生成结果中 Processing_Thread_1 明确指定了 Period、Compute_Execution_Time 和 Deadline 属性, 形成具有执行时间与响应时间约束的线程声明, 如图 13 所示.

这样的表达不仅符合 AADL 的语义要求, 也为后续调度分析、负载评估提供了必要的参数支持. 相较之下,

当移除子问题提取智能体后 (SmartGen-w/o SubTask), 模型对周期执行需求未能进行充分分解, 仅在 Process 实现体中声明调度协议, 缺乏与线程对应的具体调度参数定义, 其生成结果如图 14 所示.

```
thread Processing_Thread_1
properties
    Period => 50 ms;
    Compute_Execution_Time => 10 ms . 20 ms;
    Deadline => 30 ms;
end Processing_Thread_1;
```

图 13　线程声明

```
process implementation Task_Processing_Process.Impl
subcomponents
    Input_Acquisition_Thread:thread Input_Acquisition_Thread.Impl;
    Processing_Thread_1:thread Processing_Thread_1.Impl;
    Processing_Thread_2:thread Processing_Thread_2.Impl;
    Output_Thread:thread Output_Thread.Impl;
    Rule_Check_Thread:thread Rule_Check_Thread.Impl;
connections
    Input_To_Processing_1:port Input_Acquisition_Thread.Output -> Processing_Thread_1.Input;
    Input_To_Processing_2:port Input_Acquisition_Thread.Output -> Processing_Thread_2.Input;
    Processing_To_Output_1:port Processing_Thread_1.Output -> Output_Thread.Input;
    Processing_To_Output_2:port Processing_Thread_2.Output -> Output_Thread.Input;
properties
    Dispatch_Protocol => Periodic;
end Task_Processing_Process.Impl;
```

图 14　SmartGen-w/o SubTask 生成实例

这种写法存在多个隐患, 首先调度属性被限定在 Process 层级, 未能下传至 Thread, 使得每个线程缺乏独立的执行时间和周期定义, 编译器将报告 Period 属性未定义, 进而引发调度冲突. 其次, 模型在接口连接部分虽保留了组件拓扑, 但由于连接双方缺乏一致的周期语义, 会导致端口刷新率不一致、sender/receiver 同步失败等问题, 进一步影响系统的语义完整性与时序正确性. 此外, 在实际编译过程中, 错误不会仅集中在属性缺失的源头, 而会在所有涉及该线程的使用场景中反复触发, 从而产生大量重复性错误, 显著抬高错误率. 移除组件间连接关系识别后, 生成模型在接口连通与依赖解析层面的缺陷显著增多, 整体错误行占比和错误组件占比分别上升 4.67% 与 16.25%. 长篇需求文本通常跨越信号采集、算法处理与结果展示等多个功能域; 缺乏连接识别时, LLM 在递段式生成过程中无法维系全局拓扑一致性, 易造成端口重名、调用冲突及空引用.

实验结果进一步表明, 引入结构感知与检索增强生成机制可有效提升模型在复杂嵌入式任务中的代码生成质量. 结合任务分解与连接识别的完整流程, 显著降低了组件误配与结构冲突等高发错误, 尤其在多线程通信和功能分派场景下表现稳定. 相比之下, 缺乏结构约束的生成方式在多类系统中暴露出明显局限, 验证了辅助机制在提高系统一致性与可部署性方面的关键作用.

6.2.3　基于 DeepSeek-r1 的 SmartGen 消融实验架构语义相似度量化评估

本节主要对比了不同方案在生成结果结构一致性与语义契合度方面的表现, 采用 R_{BERT}、P_{BERT} 与 F_{BERT} 这 3 项度量指标对生成架构与参考架构之间的语义匹配程度进行建模, 实验结果如图 15 所示, 本文设计的完整生成流程 (SmartGen-AADL) 在多数系统任务中均能保持较高的结构匹配准确性. 以 F_{BERT} 值衡量其综合表现, SmartGen-AADL 在所选任务中的平均得分为 71.87%, 明显优于其简化变体 SmartGen-w/o SubTask (67.54%) 与 SmartGen-w/o Structure (66.34%), 也优于直接基于大模型生成的基线方案 LLM-direct (66.15%). 这表明, 在引入结构约束与语义细化机制后, 生成结果更符合参考架构的组织形式, 组件关系与行为模式更具一致性.

性能分布随任务复杂度而变化. 当参考架构的模块边界本身清晰 (如 Pacemaker、Time_trigger) 时, 4 种方法的差距趋于收敛, 完整流程的 F_{BERT} 仍维持在 75% 以上, 表现出稳健的泛化能力; 而在层级嵌套深、组件耦合紧的场景 (如 Radar、ROSACE), SmartGen-AADL 的 F_{BERT} 相较 LLM-direct 至少高出 13%, P_{BERT} 优势尤为明显, 说明缺乏拓扑先验时, 大模型更容易出现结构膨胀或错配.

子问题提取机制的效益通过 SmartGen-AADL 与 SmartGen-w/o SubTask 的差异得到验证: 绝大多数系统的 F_{BERT} 提升幅度保持在 5%–10% 区间. 然而在 Air_conditioner、Solar_car、Control_sys、Satellite 这 4 个系统中, SmartGen-w/o SubTask 的 R_{BERT} 略高于 SmartGen-AADL, 同时 P_{BERT} 均下滑近 10%. 检查需求文本可见, 这些场景

在最初描述阶段就存在轻微语义空缺; 缺乏细粒度任务约束后, 模型倾向引入与参考语义“相似而不一致”的细节, 从而提高覆盖度却降低匹配精度. 全局 P_{BERT} 的平均降幅 (5.5%) 大于 R_{BERT} 的提升, 验证了过度补全造成的语义漂移而非方法缺陷.

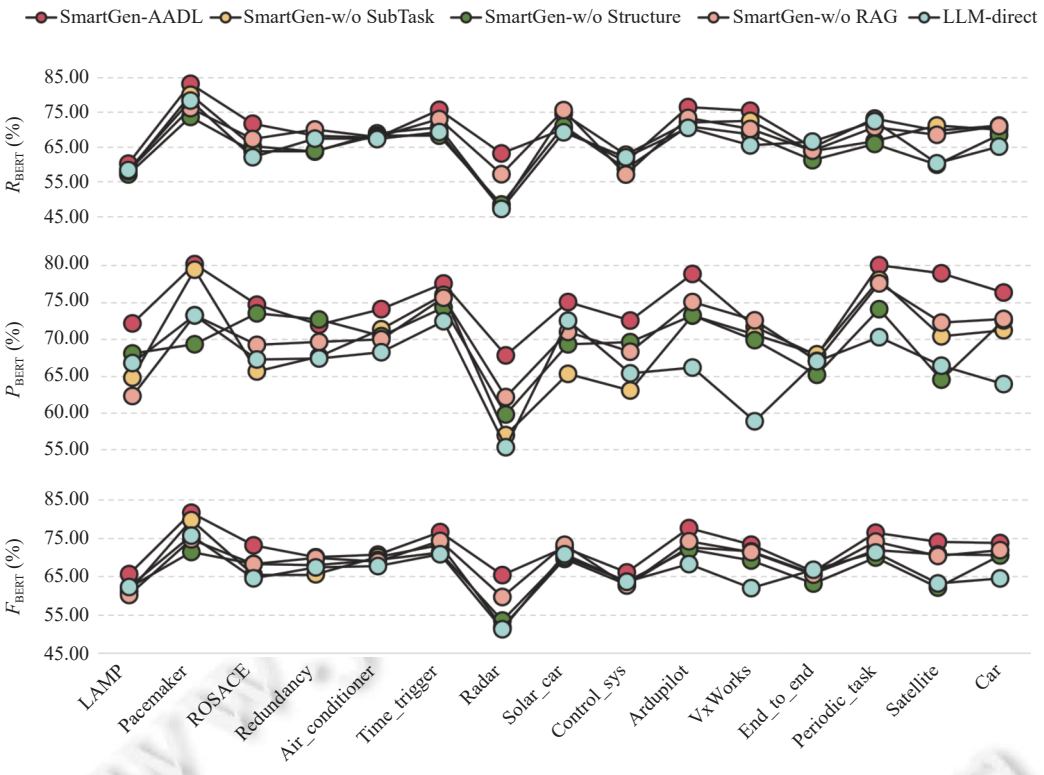


图 15 基于 DeepSeek-r1 的 SmartGen 消融实验架构语义相似度对比

系统组件结构关系识别智能体的作用由 SmartGen-AADL 与 SmartGen-w/o Structure 的差距体现. 所有系统的 R_{BERT} 与 P_{BERT} 在缺失该模块后同步下滑, 高耦合场景尤为明显: 在 Radar 中, F_{BERT} 从 65.44% 下降至 53.57%. 典型错误包括端口未绑定、信号链路中断和子程序层级缺失, 说明连接识别是维系拓扑完整性的必备环节.

整体而言, 将 RAG 与任务拆分及连接识别相结合, 可显著提高复杂 AADL 架构的语义对齐度: R_{BERT} 与 P_{BERT} 在多层级, 高耦合系统中同步受益, F_{BERT} 得分稳定高于不含任何结构提示的基线方案. 反之, 任一机制缺失都会在覆盖率或精确度维度产生系统性退化, 尤其在信号链深、组件耦合强的场景中表现最为突出.

6.2.4 基于 DeepSeek-r1 的 SmartGen 消融实验人工评分结果与分析

人工评估结果如表 11 所示, 可以看出, 本文提出的 SmartGen-AADL 方法在全部 15 个系统中获得一致领先, 其评分集中于 4.1–4.6 分区间, 平均值达 4.42 分, 显著优于其他实验配置. 相比之下, SmartGen-w/o SubTask 和 SmartGen-w/o Structure 的得分多集中于 4.0 分附近, 而 LLM-direct 则大多低于 4.0 分, 平均值仅为 3.79 分, 反映出缺乏结构引导与语义约束时, 生成模型在接口细节、组件协同与功能完整性方面难以满足实际部署要求.

评分差距的主要来源在于接口配置的准确性与系统逻辑的闭环性. SmartGen-AADL 在多个场景中均展现出优异表现, 尤其是在 LAMP 在线分析系统、Redundancy 系统与基于总线通信的端到端控制系统等结构清晰、需求链路完整的任务中, 与基线方案相比, 其综合得分提升幅度最高可达 0.96 分. 生成结果中, SmartGen-AADL 能有效规避端口重名、组件冗余与连接遗漏等结构性错误, 显著增强了模型的工程可验证性. 而在如 Radar 与 Time_trigger 等依赖复杂连接关系的系统中, 缺乏组件间连接识别机制的 SmartGen-w/o Structure 得分明显下滑, 平均落后 SmartGen-AADL 约 0.3 分, 集中表现为端口遗漏、通信路径中断与子线程调度不明确等问题, 使得评审者难以确认系统的端到端执行流程.

表 11 基于 DeepSeek-r1 的 SmartGen 消融实验人工评分结果与分析结果统计

系统	SmartGen-AADL	SmartGen-w/o SubTask	SmartGen-w/o Structure	SmartGen-w/o RAG	LLM-direct
Time_trigger	4.31	3.84	3.75	4.11	3.96
VxWorks	4.53	4.19	3.91	4.02	3.94
Radar	4.1	4.03	3.98	3.95	3.76
Car	4.55	4.4	4.15	4.39	3.89
ROSACE	4.46	4.68	4.1	4.21	3.75
Ardupilot	4.67	4.03	4.12	4.5	4.09
Pacemaker	4.53	4.22	3.82	3.98	3.97
Air_conditioner	4.18	4.2	4.11	4.17	3.43
Satellite	4.23	3.99	4.13	4.03	4.01
Solar_car	4.39	4.16	3.93	4.38	3.48
Redundancy	4.58	4.47	4.24	4.31	3.65
End_to_end	4.71	4.28	4.14	4.46	4.05
Periodic_task	4.46	4.02	3.71	4.15	3.7
LAMP	4.62	3.97	3.85	3.92	3.66
Control_sys	3.95	4.12	3.84	4.08	3.47
平均值	4.42	4.17	3.99	4.18	3.79

子问题提取机制的贡献同样通过专家评分得以体现. 在大多数系统中, SmartGen-AADL 相较 SmartGen-w/o SubTask 在综合得分上保持 0.15–0.30 分的稳定领先. 然而在 Air_conditioner、Solar_car、Control_sys 与 Satellite 这 4 个系统中, SmartGen-w/o SubTask 的得分略高于 SmartGen-AADL. 进一步分析表明, 这些需求文本在撰写阶段与参考架构之间存在轻微语义缺口, 缺乏粒度控制的 SmartGen-w/o SubTask 更倾向于在自由生成中补充合理推断性细节. 但该类补全并未带来接口准确性的同步提升, 实际在接口维度的得分仍低于 SmartGen-AADL, 反映出这些推断性细节并未完全符合参考架构的设计预期, 说明召回的提升存在精度损耗.

在去除 RAG 模块的 SmartGen-w/o RAG 实验中, 整体评分略低于完整方法, 但普遍高于端到端生成方案 (LLM-direct). 具体来看, 该设置在 Radar、Car、ROSACE 等高复杂度系统中的综合评分分别为 3.95、4.39 和 4.21 分, 接口准确性和逻辑清晰性表现明显逊色于 SmartGen-AADL. 分析原因可知, RAG 模块通过知识检索提供额外的上下文与语义补充, 有效弥补大模型在抽象需求解析、组件依赖识别及接口映射上的不足, 其缺失会导致模型在处理复杂系统时出现接口遗漏、端口冲突或逻辑链不完整等问题. 该结果进一步证明, RAG 机制在提升生成架构的工程可验证性和整体评分方面发挥了关键作用, 是保证生成高质量、可部署 AADL 架构的重要组成部分. 值得注意的是, 完整流程的评分方差明显小于其他实验配置, 表明该方案在不同任务场景下具有更强的稳定性和泛化能力. 平均而言, SmartGen-AADL 的综合评分分别比 SmartGen-w/o SubTask、SmartGen-w/o Structure、LLM-direct 高出约 0.25、0.43 和 0.63 分, 显示其在整体结构可读性、功能一致性及接口正确性方面的系统性优势. 评审者一致认为, RAG 驱动下的大模型配合结构树构建与子问题拆解机制, 能更有效地实现语义一致的组件映射、清晰的调度路径与严谨的接口定义, 从而生成具备可部署价值的高质量 AADL 架构.

总的来看, 人工评分结果与前 3 项自动化评估指标形成一致印证, 进一步确认完整流程在结构控制、语义对齐与可用性方面的综合优势. 缺乏任一关键机制的变体方案, 在不同系统任务中均表现出一定程度的语义偏差与结构失真, 尤其在高耦合或语义模糊任务中更易暴露其劣势, 难以满足工业级模型构建的质量要求.

6.3 基于 Claude-3.7 的 SmartGen 消融实验

为了分析 Claude-3.7 模型在系统架构生成中的关键模块作用, 本节采用与前一组 DeepSeek-r1 实验相同的消融实验方法. 通过逐步移除子问题提取 (SubTask)、结构树构建 (Structure) 及检索增强生成 (RAG) 模块, 并对比直接使用 LLM 生成的结果, 评估各模块对构件数量还原精度的具体影响.

6.3.1 基于 Claude-3.7 的 SmartGen 消融实验构件数量偏差度对比分析

表 12 展示了 Claude-3.7 模型在不同消融设置下各测试系统构件数量的偏差情况. 从整体来看, 完整的 SmartGen-

AADL 配置在大多数系统中能够较准确地还原构件数量, 其平均偏差度为 25.26%, 表现出较好的稳健性. 在 Time_trigger、Ardupilot 和 End_to_end 等系统中, 偏差率均低于 5%, 说明完整流程在处理中低复杂度系统时具有较高的构件数量还原精度. 移除子问题提取模块后, 偏差显著增加. 在复杂依赖系统中, 如 Control_sys 和 Solar_car, 偏差分别高达 142.86% 和 50%, 说明子问题划分对于保持构件数量与系统复杂性之间的匹配具有重要作用. 具体来看, Control_sys 系统在去掉子问题提取模块后出现 17 个构件的预测, 而原始系统仅有 7 个构件, 产生了大量冗余, 这直接导致偏差急剧上升. 移除结构树构建模块也会导致严重偏差, 尤其是在 Radar、ROSACE 和 Control_sys 系统中, Control_sys 的偏差甚至高达 271.43%, Radar 系统偏差为 47.37%, ROSACE 系统偏差达到 44.74%. 这一结果表明结构树构建在捕获系统层级关系、约束依赖及全局组合逻辑方面发挥关键作用, 缺失该模块会导致层次结构丢失和构件重复. 相比之下, 移除检索增强生成模块或直接使用大语言模型生成的结果在大多数系统中表现出更高的偏差, 平均偏差度均达到 46.81%. 例如, 在 ROSACE 和 Radar 系统中, LLM-direct 偏差分别达到 57.89% 和 47.37%, 显示检索增强生成模块在参考历史系统文档和相似组件信息, 并进行合理重排方面不可替代.

表 12　基于 Claude-3.7 的 SmartGen 消融实验构件数量偏差度对比

名称	原始构件数	指标	SmartGen-AADL	SmartGen-w/o SubTask	SmartGen-w/o Structure	SmartGen-w/o RAG	LLM-direct
Time_trigger	10	构件数	9	12	10	8	7
		偏差率 (%)	10.00	20.00	0	20.00	30.00
VxWorks	9	构件数	12	12	11	11	13
		偏差率 (%)	33.33	33.33	22.22	22.22	44.44
Radar	19	构件数	20	23	28	20	10
		偏差率 (%)	5.26	21.05	47.37	5.26	47.37
Car	15	构件数	17	16	19	17	22
		偏差率 (%)	13.33	6.67	26.67	13.33	46.67
ROSACE	38	构件数	33	34	21	32	16
		偏差率 (%)	13.16	10.53	44.74	14.79	57.89
Ardupilot	22	构件数	23	23	18	21	16
		偏差率 (%)	4.55	4.55	18.18	4.55	27.27
Pacemaker	8	构件数	9	9	11	9	8
		偏差率 (%)	12.50	12.50	37.50	12.50	0
Air_conditioner	12	构件数	14	15	17	13	11
		偏差率 (%)	16.67	25.00	41.67	8.33	8.33
Satellite	25	构件数	25	24	32	21	20
		偏差率 (%)	0	4.00	28.00	16.00	20.00
Solar_car	30	构件数	17	15	16	19	17
		偏差率 (%)	43.33	50.00	46.67	36.67	43.33
Redundancy	11	构件数	15	13	12	14	13
		偏差率 (%)	36.36	18.18	9.09	27.27	18.18
End_to_end	21	构件数	20	17	19	18	20
		偏差率 (%)	4.76	19.05	9.52	14.29	4.76
Periodic_task	9	构件数	9	9	11	9	8
		偏差率 (%)	0	0	22.22	0	11.11
LAMP	14	构件数	14	15	14	12	14
		偏差率 (%)	0	7.14	0	14.29	0
Control_sys	7	构件数	20	17	26	18	31
		偏差率 (%)	185.71	142.86	271.43	157.14	342.86
平均偏差度 (%)			25.26	24.99	41.69	24.44	46.81

此外, 部分系统出现明显异常偏差值得关注. Control_sys 系统在 LLM-direct 配置下偏差高达 342.86%, 而 Solar_car 系统在 SmartGen-w/o SubTask 和 LLM-direct 配置下偏差也均超过 40%, 说明系统复杂度和构件数量多是造成极端偏差的重要因素, 同时大语言模型在缺乏子问题划分和结构约束时容易产生漏构件或重复构件.

总体而言, 实验结果表明, 与 DeepSeek-r1 模型相比, Claude-3.7 模型在中低复杂度任务中表现稳定, 但在复杂系统中波动更大, Claude-3.7 模型尽管在平均准确性和人工评分上表现突出, 但在高层级或耦合强的系统中稳定性略显不足. 多模块协同设计能够有效提升构件数量还原精度, 尤其在复杂系统中能够避免出现极端偏差. 这为进一步优化大语言模型在系统架构自动生成中的模块化设计提供了有力的实证依据, 同时也强调了子问题提取、结构树构建和检索增强生成模块在方法中的关键作用.

6.3.2 基于 Claude-3.7 的 SmartGen 消融实验错误率指标对比分析

表 13 展示了 Claude-3.7 模型在不同消融设置下各系统的错误行占比情况. 从整体来看, 完整的 SmartGen-AADL 配置在大多数系统中的错误行占比保持在较低水平, 平均错误率为 2.33%, 说明在子问题提取、结构树构建和检索增强生成模块的协同作用下, 模型生成的系统构件与原始系统高度一致. 例如, 在 Ardupilot 和 Time_trigger 系统中, SmartGen-AADL 配置的错误率均为 0, 说明在处理低复杂度系统或结构清晰的系统时, 完整流程能够实现几乎无误的构件生成.

表 13 基于 Claude-3.7 的 SmartGen 消融实验错误行占比情况

系统	SmartGen-AADL			SmartGen-w/o SubTask			SmartGen-w/o Structure			SmartGen-w/o RAG			LLM-direct		
	错误	总数	L_{error} (%)	错误	总数	L_{error} (%)	错误	总数	L_{error} (%)	错误	总数	L_{error} (%)	错误	总数	L_{error} (%)
Time_trigger	0	302	0	6	346	1.73	8	187	4.28	8	267	3.00	12	142	8.45
VxWorks	28	230	12.17	19	214	8.88	19	209	9.09	20	197	10.15	33	269	12.27
Radar	18	651	2.76	69	810	8.52	76	1 469	5.17	20	707	2.83	40	307	13.03
Car	3	264	1.14	6	366	1.64	10	314	3.18	24	380	6.31	42	298	14.09
ROSACE	18	852	2.11	63	960	6.56	22	482	4.56	26	906	2.87	5	375	1.33
Ardupilot	0	442	0	0	501	0	9	479	1.88	15	458	3.28	23	309	7.44
Pacemaker	4	273	1.47	5	325	1.56	7	261	2.62	10	247	4.05	29	231	12.55
Air_conditioner	3	264	1.14	9	260	3.46	11	419	2.63	19	266	7.14	25	212	11.79
Satellite	10	405	2.45	15	452	3.32	30	833	3.60	29	449	6.46	100	505	19.80
Solar_car	23	713	3.23	25	671	3.73	28	492	5.69	29	659	4.40	34	245	13.88
Redundancy	3	346	0.87	10	379	2.64	8	452	1.77	16	347	4.61	59	523	11.28
End_to_end	7	467	1.50	13	435	2.99	11	489	2.25	16	546	2.93	23	285	8.07
Periodic_task	4	185	2.16	5	189	2.65	5	186	2.69	8	178	4.49	8	166	4.82
LAMP	7	527	1.33	12	587	2.04	16	575	2.78	15	546	2.75	8	186	4.30
Control_sys	11	412	2.67	13	439	2.96	21	591	3.55	20	464	4.31	57	380	15.00
平均错误率 (%)		2.33			3.51			3.72			4.64			10.54	

当移除子问题提取模块时, 错误行占比明显上升, 平均错误率达到 3.51%. 在 Radar 系统中, 错误率由 2.76% 上升至 8.52%, 而 Air_conditioner 系统的错误率从 1.14% 升至 3.46%, 显示子问题划分能够有效减少生成过程中的错误累积, 保证构件生成的正确性. 移除结构树构建模块后, 平均错误率进一步升至 3.72%, 部分系统出现较高错误率. 具体来看, Radar 系统错误率为 5.17%, Control_sys 系统错误率升至 3.55%, Solar_car 系统错误率为 5.69%, 说明结构树在维护系统层级关系和约束逻辑方面对降低错误行至关重要. 缺乏结构约束时, 模型容易出现构件层次混乱或重复构件, 导致错误行增加. 移除检索增强生成至 4.64%, 表明在缺乏历史文档和相似组件信息参考时, 模型更容易生成不一致或错误的构件. 例如, 在 Redundancy 系统中错误率从 0.87% 升至 4.61%, 而在 End_to_end 系统中错误率也升至 2.93%, 体现了检索增强生成模块在参考历史实例、实现合理重排和减少重复错误方面的重要作用. 直接使用大语言模型生成的结果表现最差, 平均错误率高达 10.54%. 部分系统错误率显著偏高, 如 Satellite 系统错误率达到 19.80%, Car 系统达到 14.09%, Control_sys 系统达到 15.00%, Solar_car 系统达到 13.88%, 这些异

常值反映出在缺乏子问题划分、结构树构建和检索增强生成的情况下, 大语言模型在复杂系统建模中容易出现漏构件、重复构件或语义错误, 导致错误行占比大幅增加.

表 14 展示了各系统错误组件占比情况, 其趋势与错误行占比一致. 完整的 SmartGen-AADL 配置平均错误组件占比为 9.97%, 保持在较低水平, 表明大部分构件生成正确. 移除子问题提取模块和结构树构建模块后, 平均错误组件占比分别上升至 16.98% 和 15.66%, 显示这两个模块在控制错误组件生成方面具有关键作用. 移除检索增强模块后错误组件占比进一步升高至 17.80%, 而直接使用大语言模型生成的错误组件占比达到 22.88%, 明显高于其他配置. 部分系统如 Redundancy、LAMP 和 Control_sys 的错误组件占比超过 30%, 说明这些系统的构件数量较多且依赖关系复杂, 当缺乏模块化辅助时, 模型生成易受累积错误影响.

表 14　基于 Claude-3.7 的 SmartGen 消融实验错误组件占比情况

系统	SmartGen-AADL			SmartGen-w/o SubTask			SmartGen-w/o Structure			SmartGen-w/o RAG			LLM-direct		
	错误	总数	C_{error} (%)	错误	总数	C_{error} (%)	错误	总数	C_{error} (%)	错误	总数	C_{error} (%)	错误	总数	C_{error} (%)
Time_trigger	0	13	0	3	16	18.75	5	14	35.71	2	11	18.18	2	10	20.00
VxWorks	7	19	36.84	7	16	43.75	2	15	13.33	5	17	29.41	7	18	38.89
Radar	4	50	8.00	12	65	18.46	10	73	13.70	6	48	12.50	4	36	11.11
Car	3	21	14.29	3	19	15.79	4	20	20.00	5	24	30.83	3	19	15.79
ROSACE	7	45	15.56	12	46	28.07	5	31	16.13	8	48	16.67	10	28	35.71
Ardupilot	0	29	0	0	30	0	3	23	13.04	4	29	13.79	2	18	11.11
Pacemaker	1	19	5.26	3	17	17.65	2	17	11.76	2	19	10.52	1	13	7.69
Air_conditioner	1	18	5.56	2	19	10.53	3	21	14.29	3	19	15.79	4	22	18.18
Satellite	4	30	13.33	6	29	20.69	8	45	17.78	7	30	23.33	4	28	14.29
Solar_car	6	59	10.17	6	47	12.76	6	39	15.38	7	51	13.73	10	37	27.03
Redundancy	2	30	6.67	4	25	16.00	2	29	6.89	5	26	19.26	7	15	46.67
End_to_end	3	31	9.71	3	26	11.54	3	30	10.00	4	27	14.81	5	25	20.00
Periodic_task	1	12	8.75	2	12	16.67	1	14	7.14	2	14	14.29	2	13	15.38
LAMP	4	32	8.75	4	32	12.50	7	31	22.58	5	30	16.67	7	25	28.00
Control_sys	2	31	6.45	3	26	11.54	6	35	17.14	5	29	17.24	9	27	33.33
平均错误率 (%)	9.97			16.98			15.66			17.80			22.88		

由表 14 的结果可知, 子问题提取、结构树构建和检索增强生成作用. 完整的多模块协同设计在复杂系统生成任务中能够保持较高准确性, 即使在构件数量多、依赖关系复杂的系统中, 也能避免异常高的错误率. 与 DeepSeek-r1 实验结果对比, Claude-3.7 模型表现出一致趋势: 完整 SmartGen-AADL 流程在大多数系统中错误率低且分布集中, 而缺失任一关键模块或直接生成方案则错误率上升且波动显著. 在高复杂度系统 (如 Radar、Car、ROSACE) 中, 结构树和 RAG 模块的缺失会显著增加错误率和错误组件占比, 表明三者协同对保障生成结果的结构正确性、接口完整性及语义一致性至关重要.

6.3.3　基于 Claude-3.7 的 SmartGen 消融实验架构语义相似度量化评估

图 16 展示了 Claude-3.7 模型在不同消融条件下各系统的 R_{BERT}、P_{BERT} 和 F_{BERT} 指标. 整体来看, 完整的 SmartGen-AADL 配置在 3 个指标上均表现出较高性能, 平均 R_{BERT} 为 71.88%, 平均 P_{BERT} 为 74.66%, 平均 F_{BERT} 为 73.14%, 显示在保持子问题划分、结构树构建和检索增强生成协同作用下, 生成系统构件与原始系统语义高度匹配, 能够较准确地反映系统功能逻辑与组件关系.

移除子问题提取模块后, R_{BERT}、P_{BERT} 和 F_{BERT} 指标均有所下降, 平均值分别为 68.94%、71.31% 和 70.23%, 尤其在复杂系统如 Control_sys、Solar_car 和 Pacemaker 中表现明显下降, 说明子问题划分在捕捉系统功能单元和保持组件间语义一致性方面具有关键作用. 移除结构树构建模块同样导致性能下降, 平均 R_{BERT} 为 70.33%, 平均 P_{BERT} 为 71.60%, 平均 F_{BERT} 为 70.90%, 部分系统如 Ardupilot 和 End_to_end 的指标下降较为明显, 反映结构树在维护系统层级关系、功能依赖和组件组合逻辑中不可替代的作用. 移除检索增强生成模块时, 平均 R_{BERT} 为 71.15%, 平均 P_{BERT} 为 72.51%, 平均 F_{BERT} 为 71.66%, 整体性能略低于完整模型, 但在部分系统如 Time_trigger、Periodic_task

仍保持较高指标, 说明 RAG 模块主要通过参考历史系统文档和相似组件信息来优化生成结果, 增强系统语义一致性. 直接使用大语言模型生成结果的平均指标明显低于 SmartGen-AADL, R_{BERT} 为 69.51%, P_{BERT} 为 70.92%, F_{BERT} 为 69.57%, 在 Ardupilot、Solar_car 和 End_to_end 等系统中下降尤为突出, 显示缺乏子问题划分、结构树和检索增强的支持, 模型难以准确捕捉系统功能逻辑和组件语义关系.

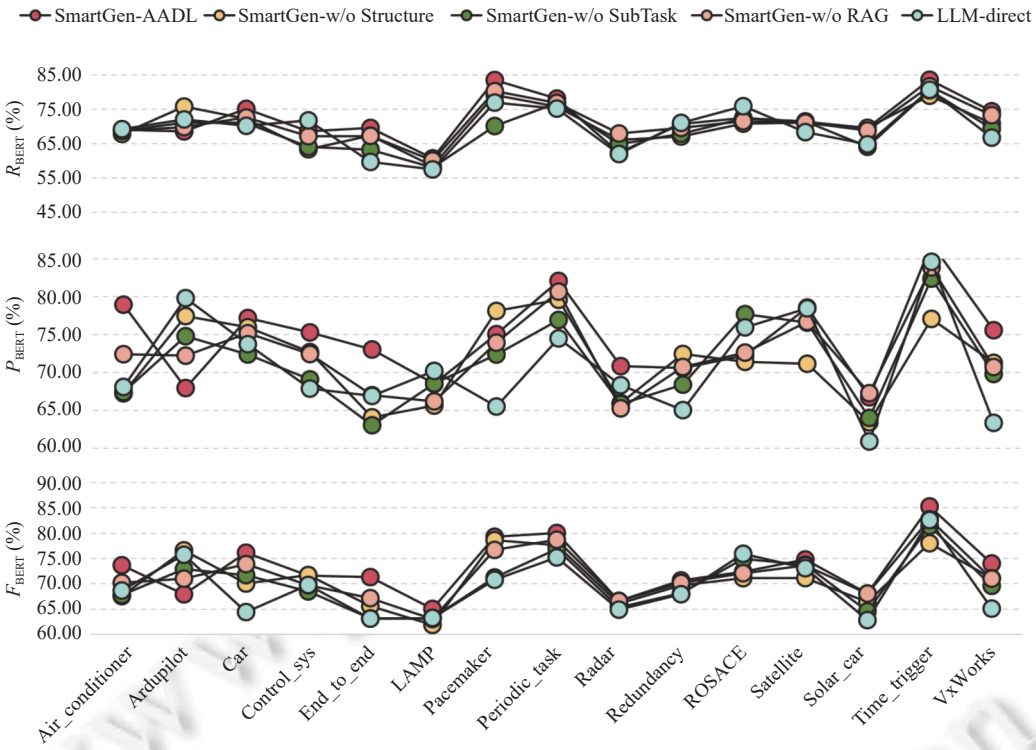


图 16 基于 Claude 的 SmartGen 消融实验架构语义相似度对比

此外, 不同系统间指标差异较大, 其中 Pacemaker、Time_trigger 和 Periodic_task 的 R_{BERT}、P_{BERT} 和 F_{BERT} 均高于平均水平, 表明这些系统的组件语义结构相对规则且易于模型捕捉. 相比之下, Solar_car、End_to_end 和 Control_sys 的指标较低, 反映系统复杂度高、依赖关系复杂, 模型在缺少模块协同时更容易产生语义偏差. 总体而言, 实验结果验证了多模块协同设计在提升构件语义匹配度和保持系统功能一致性方面的重要性, 为进一步优化 Claude-3.7 模型的模块化设计提供了实证依据.

总之, 与前述 DeepSeek-r1 实验结果相比, Claude-3.7 的表现趋势一致: 完整 SmartGen-AADL 流程在绝大多数系统中均保持更高语义匹配度, 尤其在层级嵌套深、组件耦合紧的系统 (如 Radar、ROSACE) 中, 相较 LLM-direct 的优势至少为 13%, 凸显了结构约束与模块协同的必要性. 子问题提取、结构树和 RAG 模块在不同系统任务中均展现出独立且互补的重要作用: SubTask 与 Structure 模块在接口完整性和逻辑闭环上贡献显著, 而 RAG 模块在复杂交互系统中提供关键语义与上下文补充, 三者协同确保生成架构的高质量与可部署性. 值得注意的是, 在 Pacemaker、Time_trigger 和 Periodic_task 等模块边界清晰的系统中, 各方案差距收敛, 但完整流程仍维持在 75% 以上, 显示良好的稳健性与泛化能力; 而在 Solar_car、End_to_end 和 Control_sys 等高复杂度、语义模糊系统中, 缺失任一模块都会导致语义匹配度下降, 验证了多模块协同设计在保持系统功能一致性和组件语义契合度中的重要性.

6.3.4 基于 Claude-3.7 的 SmartGen 消融实验人工评分结果与分析

人工评估结果如表 15 所示, SmartGen-AADL 在 15 个系统任务中的得分区间集中在 4.43–4.91 分, 平均值达

到 4.69, 整体表现最优且稳定性最高. 相比之下, 3 种消融设置的平均得分分别为 4.60、4.59 和 4.52 分, 而 LLM-direct 的均值仅为 4.36 分, 明显落后于完整方法. 在 Time_trigger 和 Ardupilot 等复杂系统中, SmartGen-AADL 的优势尤为突出, 分别达到 4.89 分和 4.91 分, 较 LLM-direct 提升 0.64 和 0.8 分. 特别是在 Ardupilot 系统中, SmartGen-w/o RAG 仅得分 4.62 分, 明显低于 SmartGen-AADL, 凸显了检索增强生成在高复杂度场景下的重要作用.

表 15　基于 Claude-3.7 的 SmartGen 消融实验人工评分结果与分析结果统计

系统	SmartGen-AADL	SmartGen-w/o SubTask	SmartGen-w/o Structure	SmartGen-w/o RAG	LLM-direct
Time_trigger	4.89	4.69	4.33	4.51	4.25
VxWorks	4.56	4.42	4.46	4.3	4.39
Radar	4.69	4.37	4.55	4.68	4.52
Car	4.81	4.7	4.71	4.68	4.76
ROSACE	4.59	4.53	4.58	4.71	4.32
Ardupilot	4.91	4.9	4.88	4.62	4.11
Pacemaker	4.78	4.72	4.61	4.55	4.51
Air_conditioner	4.43	4.58	4.62	4.41	4.3
Satellite	4.64	4.51	4.48	4.32	4.18
Solar_car	4.62	4.51	4.59	4.61	4.37
Redundancy	4.82	4.68	4.79	4.59	4.61
End_to_end	4.72	4.63	4.57	4.58	4.31
Periodic_task	4.53	4.59	4.62	4.39	4.13
LAMP	4.67	4.61	4.58	4.49	4.44
Control_sys	4.62	4.58	4.54	4.33	4.2
平均值	4.69	4.60	4.59	4.52	4.36

整体趋势与前文 DeepSeek-r1 模型上的实验结论一致: SubTask、Structure 和 RAG 模块在提升 AADL 架构生成质量中均发挥了关键作用. SubTask 和 Structure 模块在确保接口完整性、逻辑闭环及组件协同方面贡献突出, 而 RAG 模块在复杂交互任务中提供额外语义与上下文支持, 有助于减少端口遗漏、连接错误和逻辑断裂, 三者协同确保了生成架构的整体质量与可部署性. 与 DeepSeek-r1 的结果相比, Claude-3.7 模型下的差异更集中在接口一致性和逻辑完整性方面, SmartGen-AADL 相对基线方法的提升幅度在部分复杂系统中接近 1 分, 而在 Car、Solar_car 等系统中则差距收敛到 0.1–0.2 分, 说明其稳定性和泛化性更强.

6.4 结论分析

本研究通过设计多组消融实验, 从结构树构建、子问题拆解、检索增强生成等机制出发, 系统分析了每一模块对架构生成任务性能的实际影响. 实验结果表明, 各机制在提升系统生成的一致性、准确性与可用性方面具有显著作用, 且其贡献在不同复杂度与结构特征的任务中体现出差异化. 尤为重要的是, 消融实验的核心发现, 即“结构树构建”模块的缺失导致性能显著下降, 为当前大模型研究中的一个关键议题提供了有力的佐证: 对于形式化、结构性强的任务 (如代码生成、模型生成等), 单纯依赖 LLM 强大的隐式语言建模能力是不足够的. 我们的工作清晰地表明, 为 LLM 注入显式的结构先验或领域知识, 是引导模型生成符合严格结构规范、具备高可靠性与高质量结果的关键.

结构树构建模块作为系统的核心组成, 其主要功能在于为 LLM 提供明确的系统组织结构与层级约束, 从而引导生成结果在组件划分、端口绑定及连接路径等方面保持合理性与一致性. 实验数据显示, 该机制在大多数任务中显著降低了组件数量偏差与连接错误率, 特别是在系统规模较大、组件层级嵌套深的任务中 (如 Radar 与 ROSACE), 其结构控制能力尤为突出. 相比之下, 缺乏结构引导的生成方案易出现组件遗漏、连接断裂、端口不匹配等问题, 难以生成可部署的完整系统架构, 验证了结构先验在复杂建模任务中的必要性. 任务拆解模块则通过将复杂需求划分为若干子问题, 使模型能够以更精细的粒度理解与处理各功能单元, 有效缓解了长指令输入带来的上下文干扰问题. 该机制在任务语义边界清晰、功能划分明确的系统中表现良好, 能够提升生成结果的覆盖度与语义一致性. 然而, 在结构较为简洁或需求描述不充分的任务中, 过度拆解可能引入冗余组件, 导致生成规模膨

胀及结构偏移. 因此, 该模块的适用性与任务特征密切相关, 未来可考虑结合结构分析技术对任务进行自动可拆解性判定, 以提升鲁棒性.

检索增强生成模块通过集成领域知识库, 补充了 LLM 在 AADL 语法、建模约束及接口规则等方面的知识盲区, 显著降低了低级语法错误与属性配置错误的发生率. 该机制在涉及端口类型判定、调度属性指定与子组件语义映射等细节处理任务中表现尤为关键. 实验结果表明, 在引入检索增强生成机制后, 模型在多个任务中接口定义准确率明显提升, 专家评分中“接口一致性”与“逻辑清晰性”维度亦得到显著优化. 该结果充分说明, 语言模型虽具备较强的文本生成能力, 但在专业建模语言生成任务中, 显式的知识支撑机制不可或缺. 相比之下, 基础模型 (LLM-direct) 在不引入任何结构感知与检索增强生成机制的情况下, 整体表现不稳定, 生成结果常出现语义漂移、结构断裂及接口冲突等问题, 尤其在系统复杂度升高的任务中, 其错误率和组件偏差显著上升. 这一现象表明, 单纯依赖语言建模能力难以满足工程级建模任务中对结构完整性与语义一致性的高要求, 进一步印证了多模块协同设计对于提升生成质量的必要性.

为了与第 2 节所述基线方法进行对比, 本文基于所提出的 SmartGen-AADL 方案在 LAMP 系统上进行了建模生成实验. 针对组件 implementation Initial_State.Impl 生成结果在 features 部分的位置排布以及 calls 的语法格式上均已正确, 无明显错误, 具体生成内容如图 17 所示.

```
1:   thread Initial_State
2:   features
3:       state_change: out event port;
4:   end Initial_State;
5:   thread implementation Initial_State.Impl
6:   calls
7:       sequence: {
8:           function1: subprogram assign_random_values.Impl;
9:           function2: subprogram record_start_time.Impl;
10:       };
11:   properties
12:       Compute_Execution_Time => 2 ms . 4 ms;
13:   end Initial_State.Impl;
```

图 17 implementation Initial_State.Impl 生成结果

通过组件树结构的引导, 本文方法明确了各组件之间的调度关系与层次结构, 保障了系统整体在组织架构与时序执行上的一致性. 同时, 借助对子问题的细化分析, 进一步厘清了 LAMP 系统中各功能模块的划分, 确保职责边界清晰、功能独立. 以输入采集、任务处理、输出控制与规则校验为例, 相关功能被分别映射为独立的线程与子程序结构, 有效提升了系统模型的可用性与可维护性.

7 有效性威胁

尽管本文所提出的多智能体协同建模方法在多个典型嵌入式系统任务中取得了较为理想的实验效果, 相关指标亦验证了方法在结构建模质量与语义保持方面的有效性, 但研究过程中仍存在若干潜在威胁, 可能对结论的稳定性与外部可推广性产生影响, 主要包括以下 3 个方面.

首先, 实验所使用的数据集在规模与领域分布上均存在一定局限性. 在规模方面, 目前测试样本数量相对有限, 尚未覆盖大规模系统或超高层级的复杂嵌入式架构, 不能反映模型在极端结构条件下的适用性与稳定性. 在领域方面, 数据主要集中于航空电子、工业控制等典型场景, 尚未覆盖如医疗电子、能源系统等多个关键行业, 其系统结构、任务逻辑与语言风格可能具有显著差异, 因此模型的跨行业泛化能力尚需进一步验证.

其次, 数据集构建方法本身可能引入固有的“任务简化”偏置. 本文的数据集主要通过从已有的 AADL 模型“反向”生成自然语言需求描述来构建. 这种方式虽能高效保证需求与模型间的精准对齐, 但也导致生成的自然语言描述在结构、术语和表达上, 比真实世界中由人工撰写的需求文档更为规整和结构化, 呈现出一种“AADL 友好”的特性. 这种偏置可能带来两方面影响: 一是降低了语义解析的难度, 模型可能更多地学习到从规整文本到模型的直接映射模式, 而非真正理解模糊、多变的自然语言; 二是可能导致实验结论存在“乐观偏差”, 即模型在当前数据集上表现出的高性能, 可能无法完全复现到充满歧义、信息不全和格式不统一的真实工业需求文档中, 从而影响了对方法在真实场景下有效性的准确评估. 尽管我们已通过人工审核和多样化模板来缓解此问题, 但这一由构建方

法带来的根本性偏置仍是本研究最主要的局限性之一. 未来, 我们将持续引入真实世界的工程实例, 对数据集进行补充与迭代, 以增强其对真实应用场景的代表性.

最后, 本文采用的评估方式尽管结合了结构匹配度、语义相似性和语法正确率等定量指标, 并设置了双人交叉评审以减少人为偏差, 但在评估过程中仍不可避免地受到主观因素影响. 在任务粒度较大、结构边界模糊或存在多种合理建模路径的情形下, 不同评审者对生成结果的解读可能存在差异. 目前尚未引入如 Cohen's Kappa 系数等一致性度量方法对人工评分结果进行量化分析, 因此无法完全排除主观性干扰对实验结论稳定性造成的潜在影响. 此外, 评审专家的背景知识结构及其对嵌入式系统的理解深度亦可能影响其对模型表现的判断, 进一步加大了评估结论的波动性.

8 总 结

复杂嵌入式系统的设计正日益依赖自动化与智能化的建模技术, 以应对需求复杂性与架构约束不断增长的挑战. 本文针对自然语言需求向嵌入式系统形式化架构模型自动转换过程中存在的结构不清、组件边界模糊与交互关系缺失等问题, 提出了一种面向复杂系统的多智能体协同建模方法. 该方法融合了结构树构建、连接关系识别和知识库信息检索这 3 项关键机制, 以引导大语言模型在保持语义一致性的同时, 生成具备工程部署价值的 AADL 架构模型. 针对当前主流大模型在结构生成中的拓扑感知能力不足的问题, 本文通过多智能体解析策略与语义增强生成机制协同建模, 在多个典型嵌入式系统任务上进行了系统性消融对比实验. 实验结果表明, 与大语言模型仅依赖简单提示工程的方案相比, 所提多智能体协同建模方法在生成组件错误行占比上平均降低 34.37%, 在 F_{BERT} 语义相似度上平均提升 6.21%, 结构匹配度提升超过 20%, 并在人工评分维度上整体提高约 0.7 分. 特别是在高耦合、高层级任务中, 方法展现出更强的结构完整性控制与组件通信准确性, 能够有效减少构件重复、漏构件及语义错误, 确保生成结果在工程应用中的可用性和可靠性.

为进一步提升所提方法的通用性、稳健性与工程可迁移性, 未来工作将从以下几个维度展开: 首先, 在应用广度与评估体系方面, 将引入覆盖医疗电子、工业自动化与智能交通等领域的多层级嵌入式系统案例, 以增强模型在复杂结构和弱语义表达场景下的泛化能力. 同时, 将构建融合结构对齐、语义一致性与可部署性检查的自动化评估体系, 提升性能验证的标准化与客观性. 其次, 在协同机制层面, 本文当前采用的结构化流水线模式在处理确定性任务时效率较高, 但面对模糊或冲突性需求时灵活性不足. 因此, 未来将重点探索更复杂的智能体协同模式, 以实现从"自动化流水线"向"智能化协作平台"的演进. 具体而言: 一是引入反馈式协同机制, 构建"生成-检查-修复"的动态闭环. 例如, 当生成智能体检测到接口不匹配或逻辑错误时, 可主动请求上游的子问题分析智能体进行迭代修正, 形成具备自我完善能力的协同工作流. 二是探索协商式协同机制, 以解决智能体间的潜在冲突. 例如, 当不同模块的接口定义存在矛盾时, 可通过引入"仲裁智能体"或设计多轮协商协议, 使相关智能体达成一致, 确保最终系统模型的全局一致性. 最后, 在底层模型能力方面, 为进一步提高生成模型的语法正确率与语义一致性, 研究将结合指令微调、领域知识注入与轻量适配方法, 持续增强大模型在形式化建模任务中的适应能力与工程稳定性.

References

[1] Xu X, Ahmad E, Wang SL, Jin XY, Zhan BH, Zhan NJ. Modeling and verification of hybrid systems by extending AADL. ACM Trans. on Software Engineering and Methodology, 2025, 35(3): 1–51.[doi: 10.1145/3737698]

[2] Hallerstede S, Hatcliff J. A mechanized semantics for component-based systems in the HAMR AADL runtime. Science of Computer Programming, 2025, 245: 103312. [doi: 10.1016/j.scico.2025.103312]

[3] Li Z, Cao ZN, Wang FJ, Xing C. A modeling and verification method of cyber-physical systems based on AADL and process algebra. Int'l Journal of Software Engineering and Knowledge Engineering, 2024, 34(1): 49–89. [doi: 10.1142/S0218194023500468]

[4] Chen XY, Zhu Y, Zhao Y, Wang JY. Hybrid AADL modeling and model transformation for CPS time and space properties verification. Ruan Jian Xue Bao/Journal of Software, 2021, 32(6): 1779–1798 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/6249.htm [doi: 10.13328/j.cnki.jos.006249]

[5] Liu J, Li TF, Ding ZH, Qian YQ, Sun HY, He JF. AADL+: A simulation-based methodology for cyber-physical systems. Frontiers of Computer Science, 2019, 13(3): 516–538. [doi: 10.1007/s11704-018-7039-7]

[6] Wei XM, Dong YW, Sun C, Li XH, Ma JF. Safety analysis for mixed-criticality system with random errors and burst errors based on AADL. Ruan Jian Xue Bao/Journal of Software, 2024, 35(9): 4287–4309 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/7137.htm [doi: 10.13328/j.cnki.jos.007137]

[7] Lu Y, Qin SD, Guo P, Dong YW. Hardware-software integrated reliability modeling and analysis using AADL. Ruan Jian Xue Bao/Journal of Software, 2022, 33(8): 2995–3014 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/6610.htm [doi: 10.13328/j.cnki.jos.006610]

[8] Bardaro G, Matteucci M. Modelling robot architectures with AADL. ACM SIGAda Ada Letters, 2023, 43(1): 59–63. [doi: 10.1145/3631483.3631491]

[9] Zhao ZY, Zhang L, Lian XL, Lv HY. DRIP: Segmenting individual requirements from software requirement documents. Software: Practice and Experience, 2024, 54(5): 842–874. [doi: 10.1002/spe.3303]

[10] Carlshamre P, Sandahl K, Lindvall M, Regnell B, Dag JNO. An industrial survey of requirements interdependencies in software product release planning. In: Proc. of the 5th IEEE Int'l Symp. on Requirements Engineering. Toronto: IEEE, 2001. 84–91. [doi: 10.1109/ISRE.2001.948547]

[11] Wein S, Briggs P. A fully automated approach to requirement extraction from design documents. In: Proc. of the 2021 IEEE Aerospace Conf. (50100). Big Sky: IEEE, 2021. 1–7. [doi: 10.1109/AERO50100.2021.9438170]

[12] Kossiakoff A, Sweet WN, Seymour SJ, Biemer SM. Systems Engineering Principles and Practice. Hoboken: John Wiley & Sons Inc., 2011.

[13] Nuseibeh B, Easterbrook S. Requirements engineering: A roadmap. In: Proc. of the 2000 Conf. on the Future of Software Engineering. Limerick: ACM, 2000. 35–46. [doi: 10.1145/336512.336523]

[14] Ahmed S, Ahmed A, Eisty NU. Automatic transformation of natural to unified modeling language: A systematic review. In: Proc. of the 20th IEEE/ACIS Int'l Conf. on Software Engineering Research, Management and Applications (SERA 2022). Las Vegas: IEEE, 2022. 112–119. [doi: 10.1109/SERA54885.2022.9806783]

[15] Luttmer J, Prihodko V, Ehring D, Nagarajah A. Requirements extraction from engineering standards-systematic evaluation of extraction techniques. Procedia CIRP, 2023, 119: 794–799. [doi: 10.1016/j.procir.2023.03.125]

[16] Akay H, Yang M, Kim SG. Automating design requirement extraction from text with deep learning. In: Proc. of the 2021 Int'l Design Engineering Technical Conf. and Computers and Information in Engineering Conf. American Society of Mechanical Engineers, 2021. V03BT03A035. [doi: 10.1115/DETC2021-66898]

[17] Akay H, Kim SG. Extracting functional requirements from design documentation using machine learning. Procedia CIRP, 2021, 100: 31–36. [doi: 10.1016/j.procir.2021.05.005]

[18] Fujitake M. LayoutLLM: Large language model instruction tuning for visually rich document understanding. In: Proc. of the 2024 Joint Int'l Conf. on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024). Torino: ELRA and ICCL, 2024. 10219–10224.

[19] Zhang JY, You ZY, Wang JZ, Le XY. SAIL: Sample-centric in-context learning for document information extraction. In: Proc. of the 39th AAAI Conf. on Artificial Intelligence. Philadelphia: AAAI Press, 2025. 25868–25876. [doi: 10.1609/aaai.v39i24.34780]

[20] Wang F, Yang ZB, Huang ZQ, Zhou Y, Liu CW, Zhang WB, Xue L, Xu JM. Approach for generating AADL model based on restricted natural language requirement template. Ruan Jian Xue Bao/Journal of Software, 2018, 29(8): 2350–2370 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5530.htm [doi: 10.13328/j.cnki.jos.005530]

[21] Liu CW, Yang ZB, Zhou Y, Yuan SH, Xu JM, Xue L. Automated tool to derive AADL models from requirements in restricted natural language. Journal of Chinese Computer Systems, 2019, 40(5): 984–995 (in Chinese with English abstract). [doi: 10.3969/j.issn.1000-1220.2019.05.013]

[22] Qiu ZK, Yang ZB, Xie J, Zhou Y, Cheng GH, Chen JW. Automatic reverse construction of AADL models for safety-critical software. Journal of Chinese Computer Systems, 2022, 43(7): 1553–1561 (in Chinese with English abstract). [doi: 10.20009/j.cnki.21-1106/TP.2020-1102]

[23] Wang RL, Chen K, Shi Y. A study on transforming requirement descriptions based on the B method into AADL models. Sci-Tech & Development of Enterprise, 2023, (1): 58–60, 89 (in Chinese). [doi: 10.3969/j.issn.1674-0688.2023.01.015]

[24] Chen JL, Hu BK, Diao WW, Huang YQ. Automatic generation of SysML requirement models based on Chinese natural language requirements. In: Proc. of the 6th Int'l Conf. on Electronic Information Technology and Computer Engineering. Xiamen: ACM, 2022. 242–248. [doi: 10.1145/3573428.3573470]

[25] Wang KX, Hu J, Wang LS, Ding D. A domain specific formal modeling and analysis method for itemized requirements in avionics. In: Proc. of the 7th Int'l Conf. on Computer Information Science and Application Technology (CISAT 2024). Hangzhou: IEEE, 2024.

597–602. [doi: 10.1109/CISAT62382.2024.10695250]

[26] Zhang T, Kishore V, Wu F, Weinberger KQ, Artzi Y. BERTScore: Evaluating text generation with BERT. In: Proc. of the 8th Int'l Conf. on Learning Representations. Addis Ababa: ICLR, 2020.

[27] Papineni K, Roukos S, Ward T, Zhu WJ. BLEU: A method for automatic evaluation of machine translation. In: Proc. of the 40th Annual Meeting of the Association for Computational Linguistics. Philadelphia: ACL, 2002. 311–318. [doi: 10.3115/1073083.1073135]

附中文参考文献

[4] 陈小颖, 祝义, 赵宇, 王金永. 面向 CPS 时空性质验证的混成 AADL 建模与模型转换方法. 软件学报, 2021, 32(6): 1779–1798. http://www.jos.org.cn/1000-9825/6249.htm [doi: 10.13328/j.cnki.jos.006249]

[6] 魏晓敏, 董云卫, 孙聪, 李兴华, 马建峰. 基于 AADL 的混合关键系统随机错误与突发错误安全性分析. 软件学报, 2024, 35(9): 4287–4309. http://www.jos.org.cn/1000-9825/7137.htm [doi: 10.13328/j.cnki.jos.007137]

[7] 陆寅, 秦树东, 郭鹏, 董云卫. 软硬件综合 AADL 可靠性建模及分析方法. 软件学报, 2022, 33(8): 2995–3014. http://www.jos.org.cn/1000-9825/6610.htm [doi: 10.13328/j.cnki.jos.006610]

[20] 王飞, 杨志斌, 黄志球, 周勇, 刘承威, 章文炳, 薛垒, 许金淼. 基于限定自然语言需求模板的 AADL 模型生成方法. 软件学报, 2018, 29(8): 2350–2370. http://www.jos.org.cn/1000-9825/5530.htm [doi: 10.13328/j.cnki.jos.005530]

[21] 刘承威, 杨志斌, 周勇, 袁胜浩, 许金淼, 薛垒. 面向限定自然语言需求的 AADL 自动生成工具. 小型微型计算机系统, 2019, 40(5): 984–995. [doi: 10.3969/j.issn.1000-1220.2019.05.013]

[22] 邱志凯, 杨志斌, 谢健, 周勇, 程高辉, 陈俊文. 安全关键软件的 AADL 模型自动逆向构造方法. 小型微型计算机系统, 2022, 43(7): 1553–1561. [doi: 10.20009/j.cnki.21-1106/TP.2020-1102]

[23] 王日磊, 陈奎, 史岩. 一种基于 B 方法的需求描述转化为 AADL 模型的研究. 企业科技与发展, 2023, (1): 58–60, 89. [doi: 10.3969/j.issn.1674-0688.2023.01.015]

作者简介

葛楚妍, 博士生, CCF 学生会员, 主要研究领域为需求工程, 系统工程, 大语言模型, 智能化软件工程.

王培远, 硕士生, 主要研究领域为软件工程, 智能软件开发.

王甜甜, 博士, 副教授, 博士生导师, CCF 专业会员, 主要研究领域为软件工程, 基于模型的系统工程, 程序分析, 计算机辅助教育.

黄钇茗, 硕士生, 主要研究领域为机器学习, 软件工程.

杨小天, 本科生, CCF 学生会员, 主要研究领域为需求工程, 系统工程, 形式化验证.