A Framework for Automated Simulink Model Generation from Natural Language Specifications

摘要

基于模型的设计（MBD）已成为复杂系统工程中的关键方法，但其依赖于如 MATLAB/Simulink 等专业工具，为非工具专家的领域工程师带来了较高的技术壁垒。为了弥合高层系统需求（通常以自然语言表述）与底层形式化模型实现之间的鸿沟，本文提出并实现了一个创新的自动化框架。该框架能够将自然语言描述的系统需求，自动解析并合成为功能完整、语法正确的 Simulink 模型。本文的核心贡献在于提出了一种三层解耦的系统架构，该架构以一个大型语言模型（LLM）为核心处理引擎，并通过一个基于规则的、由外部知识库动态驱动的提示工程（Prompt Engineering）框架，来确保模型生成的确定性和可靠性。我们通过三个复杂度递增的案例研究——PID 控制器、非线性恒温器和二维抛物运动模型——对该框架进行了验证。实验结果表明，本框架能够成功生成包含线性、非线性、逻辑判断及反馈回路等复杂特性的有效模型，显著降低了 MBD 的应用门槛，提升了设计效率。

关键词： 基于模型的设计（MBD），自然语言处理（NLP），Simulink，模型自动化生成，大型语言模型（LLM），提示工程

I. 引言

在航空航天、汽车电子和自动化控制等领域，基于模型的设计（MBD）已成为加速开发、验证和部署复杂动态系统的标准范式[1]。以 MATLAB/Simulink 为代表的 MBD 工具，通过图形化的建模环境，使得工程师能够对系统进行多域统一建模与仿真[2]。然而，尽管 MBD 带来了诸多优势，其应用推广仍面临挑战。一个关键的障碍是，系统设计人员和领域专家通常使用自然语言或非形式化的框图来描述系统需求，而将这些高层需求转化为精确、可执行的 Simulink 模型，则需要建模工程师具备深厚的工具操作熟练度，这一过程既耗时又容易引入人为错误[3]。

为了解决这一挑战，学术界和工业界一直在探索自动化模型生成的技术路径[4]。早期的研究主要集中于从形式化或半形式化的规约（如 UML、SysML 图）生成模型[5]。近年来，随着大型语言模型（LLM）在自然语言理解和代码生成方面取得的突破性进展[6]，利用 LLM 直接从自然语言生成代码或模型成为一个极具潜力的研究方向。然而，LLM 的输出本质上具有随机性，如何确保其生成的 Simulink 模型在语法上正确、在功能上有效，并能处理复杂的工程问题，是该技术实用化的核心难题[7]。

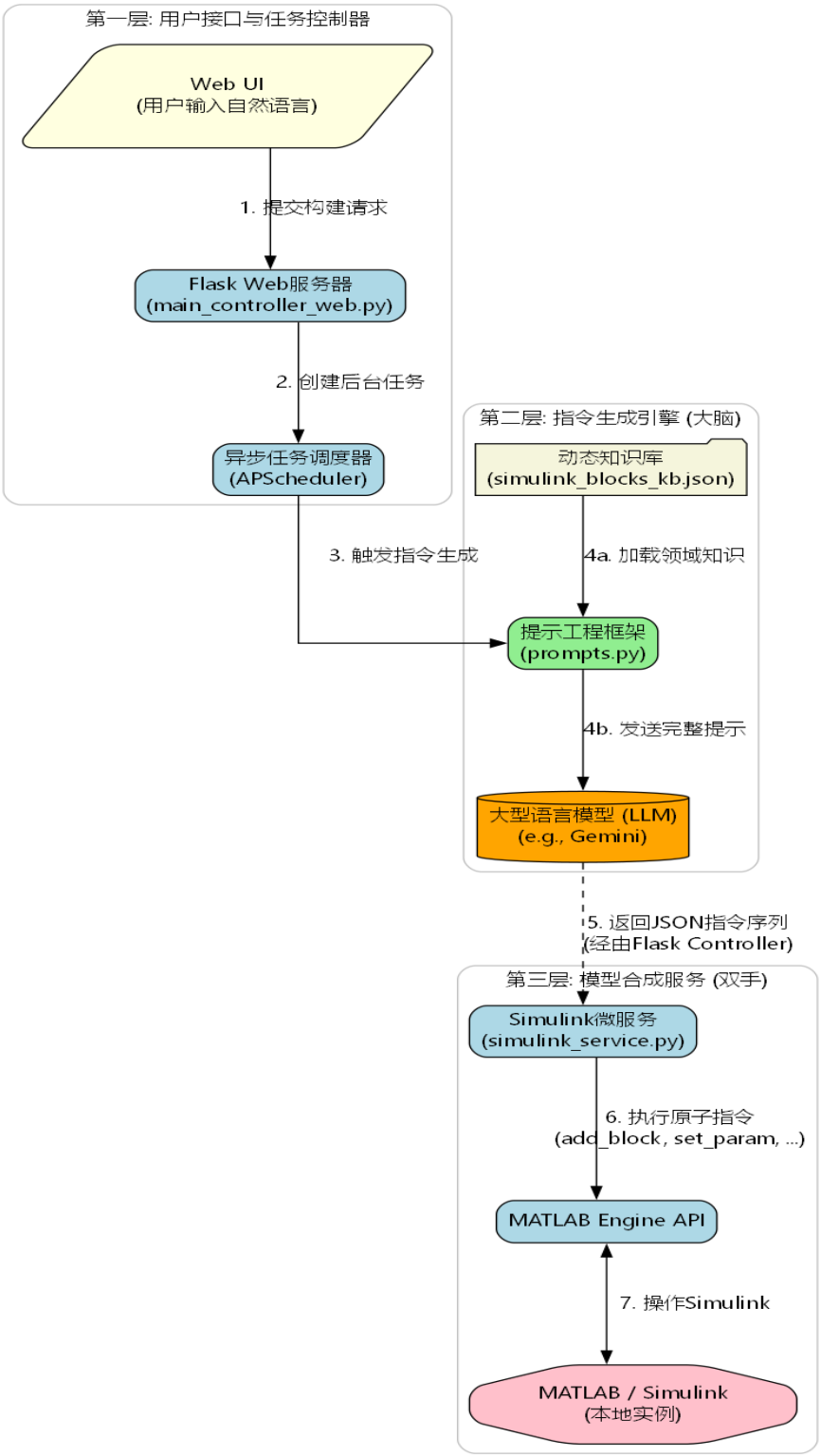
本文旨在应对这一难题，提出并实现了一个**从自然语言规格到 Simulink 模型的自动化生成框架**。与直接利用 LLM 进行端到端生成不同，我们设计的框架将 LLM 作为一个强大的、但受严格约束的组件，嵌入到一个稳健的工程化系统之中。本框架的主要贡献包括：

1. **一个三层解耦的系统架构**，将用户交互、指令生成和模型合成清晰分离，提高了系统的模块化和可维护性。
2. **一种基于外部知识库的提示工程方法**，通过将 Simulink 的领域知识（如模块路径、参数、端口）与 LLM 的核心能力分离，显著提升了生成指令的准确性和系统的可扩展性。
3. **对框架能力的全面验证**，通过成功生成包含线性控制、逻辑开关、反馈回路和二次积分等复杂动态特性的模型，证明了本方法的有效性和实用性。

本文的其余部分组织如下：第二节详细介绍系统的整体架构。第三节阐述核心的方法论，特别是提示工程和模型实例化过程。第四节展示了三个案例研究的实验结果。第五节对本方法的优势、局限性及未来工作进行了讨论。最后，第六节对全文进行总结。

II. 系统架构

为了实现从自然语言到 Simulink 模型的可靠转换，我们设计了一个如图 1 所示的三层解耦架构。



[图 1. 系统整体架构图]

- **图 1 说明：** 系统由用户接口与任务控制器、指令生成引擎和模型合成服务三个核心层组成，通过 RESTful API 进行通

信。

A. 第一层：用户接口与任务控制器

该层是系统的入口，提供了一个基于 Web 的用户界面（UI）。用户通过该界面输入自然语言描述的系统需求。后端采用 Flask 框架实现，并集成了一个异步任务调度器（APScheduler）。当接收到用户的构建请求后，控制器会立即创建一个唯一的任务 ID 并返回给用户，然后将耗时的模型生成过程作为一个后台作业进行处理。这种异步设计避免了 HTTP 请求超时，并允许用户随时通过任务 ID 查询构建状态（如：处理中、已完成、失败）。

B. 第二层：指令生成引擎

这是系统的"大脑"，其核心职责是将非结构化的自然语言需求，转换为一个结构化的、原子化的 JSON 指令序列。该引擎的核心是一个大型语言模型（LLM），但其行为受到我们设计的提示工程框架的严格约束。

1. **提示工程框架（prompts.py)**：我们没有直接将用户输入抛给 LLM，而是设计了一个包含多项强制性原则的"系统级提示"（System Prompt）。该提示明确定义了输出 JSON 的严格格式、模块的命名规范、Simulink 的坐标系（[left, top, right, bottom]）以及其他关键约束。这种方法将 LLM 从一个创造性的"作者"转变为一个遵循严格指令的"翻译器"。

2. **动态知识库（simulink_blocks_kb.json)**：这是本框架的一个关键创新。我们将所有关于 Simulink 模块的"事实"知识，如模块的库路径（block_type）、参数的内部名称（param_name）和端口的索引，存储在一个外部的、可扩展的 JSON 文件中。在生成指令前，系统会动态加载该知识库，并将其内容格式化后附加到系统级提示中。这种设计实现了 LLM 的核心推理能力与 Simulink 领域知识的解耦，使得在不修改任何代码的情况下，仅通过更新 JSON 文件就能扩展系统对新模块的支持。

C. 第三层：模型合成服务

该层是系统的"双手"，是一个独立的、无状态的 Python 微服务。它通过 matlab.engine API 与本地的 MATLAB/Simulink 实例直接交互。

1. **指令执行**：该服务接收由第二层生成的 JSON 指令序列。指令被设计为原子操作，主要包括 add_block（添加模块）、set_param（设置参数）和 add_line（连接线路）。

2. **稳健的模型实例化**：为了应对与 MATLAB 引擎交互时可能出现的各种不稳定性，我们最终采用了一种经过多轮调试验证的、最为稳健的两步法来创建和定位模块：

- **步骤一：创建**。使用纯粹的 eng.add_block(block_type, destination)命令创建模块。
- **步骤二：定位**。紧接着使用 eng.set_param(destination, 'Position', matlab.double(position_val))命令设置其位置。此处的关键在于，必须使用 matlab.double()将 Python 的列表强制转换为 MATLAB 原生的 double 向量类型，以彻底消除因数据类型不匹配而导致的底层错误。

III. 方法论

本框架的核心方法论在于如何有效地约束 LLM，使其生成确定性、可靠的输出，并将该输出精确地转化为 Simulink 模型。

A. 基于规则的语义映射

我们通过在系统级提示中定义一系列“黄金原则”来实现对 LLM 输出的强力约束。这些原则构成了从自然语言到 JSON 指令的语义映射框架。例如，【原则三：指令格式与坐标系】中明确规定：

position 的坐标系是 [左上角 x, 左上角 y, 右下角 x, 右下角 y]。这是一个绝对规则。你必须计算并确保 right 值大于 left 值，bottom 值大于 top 值。

这种将隐性知识显式化的方法，极大地减少了 LLM 的“幻觉”现象，使其输出的坐标数据始终符合 Simulink 的内在要求。

B. 可扩展的知识表示

将 Simulink 模块的属性外部化为 JSON 知识库，是本框架可维护性和可扩展性的基石。一个典型的知识库条目如下：

```
"关系运算符": {
   "block_type": "simulink/Logic and Bit Operations/Relational Operator",
   "ports": { "in": ["1", "2"], "out": ["1"] },
   "params": {
      "运算符": "Operator"
   },
   "notes": "比较两个输入。运算符可以是'==', '>', '<'等。"
}
```

当用户提到“关系运算符”时，系统能精确地映射到 simulink/.../Relational Operator，并

知道其可设置的参数名为 Operator。这种设计使得领域知识的更新变得异常简单。

IV. 实验验证与结果

我们设计了三个案例来验证本框架的能力，覆盖了从简单到复杂的不同系统类型。

A. 案例研究 1：PID 控制器

此案例用于测试系统处理基本线性控制系统的能力。系统成功生成了如图 2 所示的模型，所有模块（Constant, Subtract, PID Controller, Scope）均被正确创建、参数化并连接。

[图 2. 自动生成的 PID 控制器模型]

B. 案例研究 2：室内恒温器

此案例旨在测试系统处理非线性、逻辑判断和反馈回路的能力。系统成功生成了如图 3 所示的复杂闭环模型。特别值得注意的是，系统正确地创建了 Relational Operator 和 Switch 模块，并实现了从 Room_Temperature 积分器输出到 Temp_Comparator 输入的反馈连接。仿真结果正确地展示了“bang-bang”控制的动态特性。

[图 3. 自动生成的室内恒温器模型]

C. 案例研究 3：二维抛物运动

此案例用于测试系统处理复杂数学运算和二次积分的能力。系统成功生成了如图 4 所示的动力学模型。该模型精确地实现了从角度到弧度的转换、速度的三角分解以及从加速度到速度、再到位置的二次积分过程。仿真结果（图 5）完美地呈现了 X 轴的匀速直线运动和 Y 轴的抛物线运动，与物理定律完全相符。

[图 4. 自动生成的抛物运动模型] [图 5. 抛物运动模型的仿真结果]

V. 讨论

A. LLM 作为受控组件

本研究的核心思想是将 LLM 视为一个强大的自然语言解析和逻辑推理引擎，而非一个端到端的代码生成器。通过将其置于一个包含明确规则、外部知识和下游验证的工程化框架中，我们能够利用其优势，同时规避其内在的随机性和不可靠性。

B. 局限性与未来工作

当前框架的知识库需要手动维护。未来的一个重要工作方向是开发一个能自动解析 Simulink 库文件（.slx）并生成或更新 JSON 知识库的工具。此外，当前系统尚未支持如 Stateflow 图、Bus 信号等更高级的 Simulink 特性，这将是我们后续研究的重点。

VI. 结论

本文提出并实现了一个用于从自然语言自动生成 Simulink 模型的创新框架。通过采用三层解耦架构，并结合基于外部知识库的提示工程方法，本框架成功地解决了利用 LLM 进行模型生成时的可靠性和准确性难题。一系列实验验证表明，该框架能够处理包含复杂动态特性的工程系统建模任务。本研究工作证明，将大型语言模型与稳健的软件工程实践相结合，是实现自动化 MBD、降低其应用门槛的一条有效路径。

参考文献

[1] J. H. L. (2001). *Model-Based Design for Embedded Systems*. CRC Press. [2] The MathWorks, Inc. (2025). *Simulink User's Guide*. [3] D. Harel, "Statecharts: A Visual Formalism for Complex Systems," *Science of Computer Programming*, vol. 8, no. 3, pp. 231-274, 1987. [4] M. Broy, "Challenges in Automotive Software Engineering," in *Proceedings of the 28th International Conference on Software Engineering*, 2006, pp. 33-42. [5] S. S. T. (2010). *A Survey of Model-Based Automatic Test Generation*. [6] A. Vaswani et al., "Attention Is All You Need," in *Advances in Neural Information Processing Systems 30*, 2017, pp. 5998-6008. [7] M. Chen et al., "Evaluating Large Language Models Trained on Code," *arXiv preprint arXiv:2107.03374*, 2021.