面向单元、集成测试的智能需求约束提取的方案

研究计划草案

2025 年 7 月 15 日

目录

1 研究内容

1.1 面向单元测试的需求约束智能提取方法

确保单元测试的有效性与可信度，其核心挑战在于如何将自然语言需求的非形式化描述，精确转化为可对代码行为进行刚性验证的形式化断言。软件的功能正确性最终体现在代码层面，但需求的初始形态却是模糊且多样的自然语言。然而，现有方法普遍采用黑盒式的序列到序列（Seq2Seq）生成范式，将需求直接映射为测试代码。这种范式绕过了对需求进行结构化解析与逻辑形式化的关键步骤，其生成过程缺乏与代码库事实（如类定义、方法签名）的交互验证，极易导致模型在开放式生成中产生“事实性幻觉”，即生成的测试断言与代码的实际能力或业务逻辑不符，从而严重削弱了测试的可靠性与有效性，难以满足工业级军用软件验证的严苛需求。

为此，本项目拟构建一个代码上下文感知的需求约束分层解析与验证框架。首先，利用代码库的抽象语法树（AST）与函数摘要作为“基准真相”，引导大语言模型将原始需求文本解析为一种中间语义表示（Intermediate Semantic Representation, ISR），捕捉核心的“实体-行为-条件”逻辑。其次，通过建立需求 ISR 中的概念实体与代码 AST 中具体节点的“锚定”关系，实现从文本语义到代码实体的精准链接。最后，将“需求文本片段 → 语义表示 → 代码节点 → 可执行断言”的层级化对齐作为约束条件，构建一个可追溯、可验证的约束表征，以支撑高保真度的测试断言智能生成。

1.2 面向集成测试的需求约束序列智能构建方法

实现有效的系统集成测试，关键在于如何从离散、非形式化的需求文本中自动综合出能够体现复杂业务逻辑的测试序列。系统的整体功能由多个相互关联的子功能协同完成，其正确的交互逻辑往往隐含在分散的需求描述之中，形成巨大的“知识鸿沟”。然而，现有方法或依赖于耗时且易过时的手动建模，或局限于无法感知业务逻辑的代码结构覆盖。而新兴的大语言模型虽能生成流畅的场景描述，但其本质上是基于概率的文本补全，倾向于生成线性的标准流程，难以自主推理和构建跨越多个需求的、包含分支与异常处理的复杂交互流程，缺乏对系统行为的全局一致性理解，无法保证关键业务场景的测试覆盖完备性。

为此，本项目拟提出一个面向业务逻辑的跨需求依赖推理与测试序列综合框架。首先，利用大语言模型从每个独立的需求描述中，识别并提取出细粒度的“业务事件”单元。其次，引入神经符号推理机制，引导模型在整个需求集范围内，对提取的业务事件进行因果、时序及数据流的依赖关系推理，从而将隐式的业务流程显式化，构建成一个“跨需求依赖图”。最后，在该依赖图上进行符合业务逻辑约束的路径规划与综合，构建一个动态且完备的系统行为模型，以支撑复杂测试序列的智能化、自动化生成。

2 关键科学问题

精准且可验证的需求-代码对齐是实现高可信自动化软件测试的前提。在航天工程软件中，自然语言需求常以形式如“姿态控制系统在轨飞行期间需每秒更新一次控制指令”进行描述，而对应的可执行测试逻辑则表现为 `assert(updateRate == 1Hz && mode == OrbitFlight)`。这种从抽象语义到具体代码的映射过程存在显著的层次鸿沟与语义差异。现有基于大语言模型的端到端生成方法，试图以“扁平化”的序列映射方式直接跨越此鸿沟，却往往忽略了需求内部隐含的复杂逻辑约束与代码世界的结构化实体之间的对应关系，导致生成的测试缺乏逻辑严谨性和可追溯性，严重影响了自动化测试在关键任务软件中的可靠度。因此，如何融合大语言模型的上下文理解能力与代码库的结构化先验知识，深度挖掘并对齐需求子句、代码路径与测试断言之间的跨模态、层级化映射关系，构建一个语义忠实于需求、逻辑绑定于代码的自动化测试生成模型，是本项目拟解决的关键科学问题。

3 研究方案

3.1 代码上下文感知的需求约束分层解析与验证方法

通用的大语言模型在处理代码生成任务时，多采用开放式的序列到序列生成范式。但将其直接应用于根据需求生成单元测试时，因其忽略了软件代码的严谨结构与程序逻辑的内在约束而表现不佳。此外，自然语言需求与测试断言之间存在复杂的“一对多”或“多对一”的层级化映射关系，现有“文本到代码”的扁平化生成方法难以有效建模这种深层关联，常导致生成的测试用例缺乏可追溯性与逻辑可靠性。

为此，本项目拟提出一种代码上下文感知的需求约束分层解析与验证方法，旨在建立需求文本与测试断言之间精准、可解释的生成路径。首先，我们将航天任务软件的代码库中提取的抽象语法树（AST）与关键功能模块的行为摘要作为“基准真相”注入大语言模型，引导其将原始自然语言需求解析为包含“实体-行为-条件”等核心要素的中间语义表示（Intermediate Semantic Representation, ISR），从而为模型注入具有航天软件领域特征的结构化知识。其次，设计跨模态实体锚定机制，在需求 ISR 与代码 AST 之间建立精确的对应关系，将需求描述中的抽象概念（如“姿态角速率需保持在安全阈值内”）与代码实现中的具体变量（如 `gyro.angularRate`）进行双向绑定，确保生成的测试逻辑既具语义一致性，又能与程序实体精确对齐。

随后，构建一个层级化的受控生成模型，该模型以“需求片段 → ISR → 代码实体”的对齐关系为强约束，将测试生成任务转化为从形式化逻辑到测试断言的翻译过程，极大降低“模型幻觉”的风险。最后，通过执行生成的测试并分析其结果，将编译错误、运行时异常或断言失败等信息作为反馈信号，以闭环方式迭代优化整个框架，确保生成的测试断言在实际工程中的有效性。图 **X** 为本方法的技术框架示意图。（内容确认后，

加入图片）

3.1.1 基于代码库知识增强的需求语义解析

该步骤的核心目标是构建一个能深度理解软件开发上下文的语义解析模型，将非结构化的自然语言需求 R 精准地转化为结构化的中间语义表示（Intermediate Semantic Representation, ISR），记为 Y_{ISR}。此过程并非孤立的文本分析，而是深度融合了目标代码库的先验知识。

首先，我们将对目标代码库进行预处理，构建一个多模态的“代码上下文知识库” $\mathbf{C}$。具体实现上，我们会利用静态分析工具提取代码的抽象语法树（AST），并采用图神经网络（GNN）来学习其结构化信息，得到每个代码节点 v 的嵌入表示 $\mathbf{h}_v = \text{GNN}(AST)_v$。同时，我们从代码中抽取出 API 文档、函数注释等文本信息，利用预训练语言模型（如 BERT）将其编码为对应的文本嵌入。这两部分知识将被融合，构成一个全面的代码上下文表征，其数学表达为：

$$\mathbf{C} = \text{Fusion}(\{\mathbf{h}_v\}_{v \in AST}, \text{BERT}(\text{Docs})) \quad (1)$$

此处的 Fusion 函数可以是一个简单的拼接操作或更复杂的注意力融合网络。

其次，我们将设计一个基于 Transformer 架构的、具备代码上下文感知能力的序列到序列解析模型。该模型以自然语言需求 R 为输入，其编码器（Encoder）负责生成需求的初始语义向量。在解码器（Decoder）生成 Y_{ISR} 的每一个步骤中，我们创新性地引入一个跨模态注意力模块，使解码器不仅关注需求本身，还能动态地从代码上下文知识库 $\mathbf{C}$ 中检索并聚焦于最相关的代码知识。模型的训练目标是最大化在给定需求和代码上下文的条件下，生成正确 Y_{ISR} 的条件概率，其损失函数 $\mathcal{L}_{parse}$ 定义为标准的负对数似然：

$$\mathcal{L}_{parse} = -\sum_{t=1}^{|Y_{ISR}|} \log P(y_t | y_{<t}, R, \mathbf{C}; \theta) \quad (2)$$

通过优化此目标函数，模型能够学习到一种从自然语言到半结构化、且与代码实现紧密相关的 ISR 的映射函数 $f_\theta : (R, \mathbf{C}) \rightarrow Y_{ISR}$。

3.1.2 需求语义与代码实体的跨模态锚定

在获得结构化的需求语义表示（ISR）后，本步骤旨在建立 ISR 中抽象的业务实体与代码库中具体的程序实体（如变量、方法、类）之间明确的对应关系，即“跨模态锚定”。这一步骤是连接需求与代码、确保测试可追溯性的关键环节。

首先，我们将设计一个高效的候选实体检索机制。对于 ISR 中识别出的每一个业务实体 e_{isr}（例如，从“调整订单价格”中识别出的“订单价格”），我们将基于词法和语义相似性，从代码库的 AST 节点集合 V_{AST} 中召回一个候选子集 $Cand(e_{isr})$。词法层

面，我们使用 BM25 等稀疏检索算法匹配代码标识符；语义层面，我们利用预训练的代码语言模型（如 CodeBERT）计算 e_{isr} 与所有代码实体标识符之间的语义相似度，筛选出 Top-K 个候选者。

其次，为了从候选集中精准地选出最佳匹配，我们将构建一个专门用于排序的跨模态双编码器（Dual Encoder）模型。该模型包含两个独立的 Transformer 编码器，f_{ISR} 和 f_{Code}，分别用于将 ISR 实体和候选代码实体的上下文信息（如代码片段、注释）编码为高维向量。两个实体间的匹配度通过它们向量表征的余弦相似度来度量：

$$S(e_{isr}, v_j) = \frac{f_{ISR}(e_{isr}) \cdot f_{Code}(v_j)}{\|f_{ISR}(e_{isr})\| \|f_{Code}(v_j)\|} \quad (3)$$

其中 $v_j \in Cand(e_{isr})$。最终，得分最高的候选者 v^* 将被选定为该 ISR 实体的“锚点”：

$$v^* = \arg \max_{v_j \in Cand(e_{isr})} S(e_{isr}, v_j) \quad (4)$$

为了训练这两个编码器，我们将采用对比学习范式，其损失函数（InfoNCE Loss）旨在拉近正样本对（正确的 ISR-代码匹配）的距离，同时推远负样本对的距离：

$$\mathcal{L}_{anchor} = -\log \frac{\exp(S(e_{isr}, v^+)/\tau)}{\exp(S(e_{isr}, v^+)/\tau) + \sum_{v^- \in Neg} \exp(S(e_{isr}, v^-)/\tau)} \quad (5)$$

其中 v^+ 是正样本，v^- 是从负样本集合 Neg 中采样的负样本，τ 是温度超参数。通过此步骤，我们将得到一个用代码实体“标注”过的、语义丰富且接地气的“锚定后 ISR”。

3.1.3 层级化约束引导的测试断言生成

此步骤的目标是基于前序步骤产出的、与代码实体严格锚定的中间语义表示（ISR），以一种高度受控的方式智能生成可执行的单元测试断言（Test Assertion）代码。这从根本上区别于传统的、易产生幻觉的开放式代码生成。

首先，我们将构建一个“锚定语义图”G_{AISR} 作为生成模型的直接输入。该图以 ISR 的逻辑结构为主干，其中代表业务实体的节点已通过步骤二被替换为指向代码 AST 中具体节点的指针。例如，一个逻辑表达式 ‘greaterThan(order.amount, 1000)‘ 将被表示为一个图结构，其中 ‘order.amount‘节点直接链接到代码中对应的变量。这种图结构输入确保了生成过程的所有信息都源自于经过验证和锚定的语义，而非模型的凭空想象。

其次，我们将设计一个语法约束下的层级化解码器。该解码器基于 Transformer 架构，但在生成测试断言代码的每一个时间步 t，其输出词汇表的概率分布将受到目标编程语言（如 Java、Python）的上下文无关文法（CFG）的严格约束。具体而言，我们会预先构建一个语法状态机，在每个解码步骤，只有符合语法规则的下一个词元（token）

集合 $V_{valid,t}$ 才会被考虑。模型的输出概率将进行如下的掩码操作：

$$P(a_t|a_{<t}, G_{AISR}) \propto \begin{cases} \exp(\text{score}(a_t)) & \text{if } a_t \in V_{valid,t} \\ 0 & \text{otherwise} \end{cases} \quad (6)$$

其中 a_t 是待生成的词元，score(a_t) 是模型输出的原始 logit 值。这种方式保证了生成的代码片段在语法层面是绝对正确的。模型的训练目标是学习一个从锚定语义图到代码断言的翻译函数 $T_\theta : G_{AISR} \rightarrow \text{Code}_{Assert}$。其损失函数将包含两部分，一是标准的交叉熵损失 $\mathcal{L}_{CE}$，二是“锚点利用损失” $\mathcal{L}_{util}$，用于惩罚模型未能正确使用输入图中给定的代码实体锚点：

$$\mathcal{L}_{gen} = \mathcal{L}_{CE}(A_{gen}, A_{gt}) + \lambda \mathcal{L}_{util}(A_{gen}, G_{AISR}) \quad (7)$$

其中 A_{gen} 和 A_{gt} 分别是生成的和真实的断言，λ 是权重系数。通过此方案，我们将实现一个从需求到断言的、高度可控且可解释的生成过程。

3.1.4 基于执行反馈的闭环优化与验证

为了确保模型生成的测试断言不仅在理论上合理，更在实际工程中有效，本步骤将构建一个自动化的闭环反馈系统，利用测试执行的真实结果来迭代式地优化和验证整个生成框架。此过程将引入强化学习（Reinforcement Learning, RL）的思想。

首先，我们将搭建一个自动化的测试执行与反馈收集流水线。对于模型生成的每一个测试断言 A，该流水线会自动将其注入到项目的测试框架中，并执行测试。执行过程中，系统会收集一系列结构化的反馈信号，包括：1）编译状态 $R_{compile}$（二进制奖励，成功为 1，失败为-1）；2）执行状态 R_{exec}（二进制奖励，正常结束为 1，抛出未捕获异常为-1）；3）验证价值 R_{value}，我们创新性地将其定义为测试断言对代码覆盖率（如行覆盖、分支覆盖）的贡献度，一个能覆盖新代码的断言被认为更具价值。最终，这些信号将被加权组合成一个综合的奖励函数：

$$R(A) = w_c R_{compile}(A) + w_e R_{exec}(A) + w_v R_{value}(A) \quad (8)$$

其中 w_c, w_e, w_v 是超参数权重，用于平衡不同反馈的重要性。

其次，我们将整个需求解析到断言生成的端到端模型视为一个策略网络 π_θ，它在给定的需求和代码库下，随机生成一个测试断言动作 A。我们将采用策略梯度算法（如 REINFORCE 或 PPO）来优化此策略。根据策略梯度定理，模型参数 θ 的更新梯度与期望奖励直接相关：

$$\nabla_\theta J(\theta) = \mathbb{E}_{A \sim \pi_\theta}[R(A) \nabla_\theta \log \pi_\theta(A|R, \mathbf{C})] \quad (9)$$

此公式的直观含义是：提高能获得高奖励的断言的生成概率，同时降低导致低奖励（如

编译失败）的断言的生成概率。为了兼顾训练稳定性与探索效率，我们最终的优化目标将是监督学习损失与强化学习目标的加权组合：

$$\mathcal{L}_{final} = (1-\alpha)\mathcal{L}_{gen} - \alpha J_{RL}(\theta) \quad (10)$$

其中 $\mathcal{L}_{gen}$ 是步骤三中的监督损失，α 是平衡因子。通过这个闭环系统，模型将不断自我演进，学习生成更高质量、更能发现潜在问题的测试断言。

3.2 面向业务逻辑的跨需求依赖推理与测试序列综合框架

传统的集成测试用例生成方法通常依赖于手动的业务流程建模或基于代码结构的覆盖率分析，但直接应用大语言模型进行端到端生成时，因其难以理解和推理散落在不同需求文本中的隐式交互逻辑而表现不佳。此外，复杂的业务流程往往包含大量的分支、并发和异常路径，与需求文本的线性描述之间存在复杂的“多对多”依赖关系，现有方法难以有效综合出完备的测试序列。

为此，本项目拟提出一种面向业务逻辑的跨需求依赖推理与测试序列综合方法，旨在实现从离散需求到高覆盖率、逻辑连贯的集成测试序列的自动化构建。首先，我们将设计一种细粒度的“业务事件”提取框架，利用大语言模型将每个独立、非结构化的需求文本解析为包含 $\langle$ 主体, 行为, 客体, 条件 $\rangle$ 等要素的原子化、标准化事件单元。其次，引入神经符号推理机制，构建跨需求依赖图（Cross-Requirement Dependency Graph），该模型以提取的业务事件为节点，利用 LLM 的语义理解能力和符号逻辑的推理能力，自动挖掘并建立事件之间隐含的因果、时序和数据流依赖关系，从而将隐性的业务全景显式化。

随后，我们将在构建的依赖图上，研究基于约束满足的图路径规划算法，该算法不仅能生成代表“标准流程”的主路径，更能系统性地探索并综合出覆盖多种“替代路径”与“异常路径”的测试场景。最后，通过搭建自动化执行环境，运行生成的测试序列并收集代码覆盖率、接口调用序列等反馈信息，以闭环方式对依赖推理的准确性和路径规划的有效性进行迭代优化，持续提升测试序列的缺陷发现能力。图 **Y** 为本方法的技术框架示意图。**(**内容确认后，加入图片**)**

3.2.1 面向业务流程的细粒度事件提取

该步骤旨在将宏观、非形式化的自然语言需求 R_i 分解为结构化、标准化的“业务事件”（Business Event）单元 E，为后续的依赖推理和流程构建提供原子化的输入。一个业务事件被形式化地定义为一个四元组：

$$E = (s, a, o, c) \quad (1)$$

其中，s 是事件的执行主体（Subject），a 是核心行为（Action），o 是行为的客体（Object），c 是事件发生的条件或约束（Condition）。

首先，我们将定义一套详细的事件标注规范，并构建一个小规模、高质量的“黄金”标注数据集 $\mathcal{D}_{annotated}$。我们将邀请领域专家对数百条真实需求进行手工标注，将自然语言句子映射为结构化的事件序列。例如，对于需求“VIP 用户登录后，系统应自动发放优惠券”，它将被解析为两个事件：E_1=(‘用户‘,‘登录‘,‘系统‘,‘用户是 VIP‘) 和 E_2=(‘系统‘,‘发放‘,‘优惠券‘,‘登录成功后‘)。

其次，我们将采用基于预训练大语言模型（如 Llama, T5）的序列到序列学习框架来训练事件提取器。与直接生成文本不同，该模型被训练用于将输入的需求句子 R_i 翻译成一个线性的、结构化的事件描述文本 Y_{event}。模型的训练目标是最大化在给定需求下，生成正确事件序列的条件概率：

$$P(Y_{event}|R_i;\theta) = \prod_{t=1}^{|Y_{event}|} P(y_t|y_{<t}, R_i;\theta) \quad (2)$$

我们将利用构建的黄金数据集 $\mathcal{D}_{annotated}$ 对模型进行微调，其损失函数定义为：

$$\mathcal{L}_{extract} = -\frac{1}{|\mathcal{D}_{annotated}|} \sum_{(R_i,Y^*_{event})\in\mathcal{D}_{annotated}} \log P(Y^*_{event}|R_i;\theta) \quad (3)$$

其中 Y^*_{event} 是专家标注的基准真相。通过这种方式，模型将学会从多样的自然语言表达中鲁棒地抽取出标准化的业务事件，为后续构建全局业务视图奠定坚实基础。

3.2.2 跨需求事件的因果与时序依赖推理

在提取出离散的业务事件集合 $V_E = \{E_1, E_2, ..., E_n\}$ 后，本步骤的核心任务是挖掘并建立这些事件之间隐含的逻辑依赖关系，构建一个全局的“跨需求依赖图” $G = (V_E, E_D)$。我们定义了三种关键的依赖关系类型 $d \in$ {ENABLES, PRECEDES, USES}，分别代表因果、时序和数据流依赖。

首先，我们将设计一个基于大语言模型的成对事件依赖分类器。对于任意一对从需求集合中提取的事件 (E_i, E_j)，我们将它们的文本描述以及原始上下文信息拼接起来，形成一个详细的提示（Prompt），输入给一个经过特殊微调的大语言模型。该模型被训练用于预测这对事件之间最可能存在的依赖关系类型 d。其输出是一个概率分布：

$$P(d|E_i, E_j) = \text{softmax}(\mathbf{W} \cdot \text{LLM-Encoder}([E_i; E_j])) \quad (4)$$

其中，$\mathbf{W}$ 是一个可学习的分类头。我们将基于预测概率超过某一阈值的关系，构建一个初步的、可能包含噪声的依赖图 G'。

其次，为了保证最终依赖图的逻辑一致性和完备性，我们将引入符号逻辑推理对初

步图 G' 进行修正和增强。我们将预先定义一组逻辑规则公理，例如时序关系的传递性，可以用一阶逻辑表示为：

$$\forall E_i, E_j, E_k \in V_E : \\ (\text{PRECEDES}(E_i, E_j) \land \text{PRECEDES}(E_j, E_k)) \implies \text{PRECEDES}(E_i, E_k) \quad (5)$$

我们将利用逻辑推理机（如 Answer Set Programming 求解器）或基于图的算法，检查并执行这些规则。这个过程会增加由传递性等关系推导出的新边，并删除与公理相矛盾的边（例如，循环的时序依赖），从而得到一个逻辑自洽的最终依赖图 $G = (V_E, E_D)$。图中的每一条边 $e_{ij} \in E_D$ 都是一个三元组：

$$e_{ij} = (E_i, E_j, d_{ij}) \quad (6)$$

其中 d_{ij} 是经过推理验证的依赖类型。这个图构成了后续测试序列生成的基础。

3.2.3 基于依赖图的多路径测试序列规划与综合

获得全局的跨需求依赖图 G 后，本步骤旨在从图中系统性地规划和生成一组能够全面覆盖复杂业务逻辑的测试序列（路径）集合 $\mathcal{P} = \{P_1, P_2, ...\}$。

首先，我们将对依赖图进行拓扑分析，识别出流程的起点和终点。通常，图中没有任何 ‘PRECEDES‘前驱的节点可作为潜在的起始事件，没有任何 ‘PRECEDES‘后继的节点可作为潜在的终止事件。一个有效的测试序列 $P = \langle E_1, E_2, ..., E_k \rangle$ 必须是一个从起点到终点的、满足所有依赖约束的事件序列。路径的有效性约束可以形式化地表述为：路径中的任意事件 E_k，其所有 ‘ENABLES‘类型的依赖前驱必须已经出现在该路径的前半部分 $\langle E_1, ..., E_{k-1} \rangle$ 中：

$$\forall k \in [2, |P|], \forall (E_j, E_k, \text{ENABLES}) \in E_D \implies \exists j' < k \text{ s.t. } E_{j'} = E_j \quad (7)$$

其次，为了生成多样化的测试路径，我们将设计一种基于蒙特卡洛树搜索（MCTS）的路径规划算法。与传统的确定性搜索不同，MCTS 通过模拟和随机抽样来探索图空间，能够有效地发现非最优但同样有效的高价值路径，从而覆盖标准流程之外的替代流和边缘场景。算法在每一步会评估当前路径的价值，其价值函数 $Score(P_{partial})$ 将综合考虑路径长度、新覆盖的节点/边数量等因素，以激励对更广阔图空间的探索：

$$Score(P_{partial}) = w_l \cdot \text{length}(P_{partial}) + w_c \cdot \text{coverage}(P_{partial}) \quad (8)$$

最终，我们的目标是生成一个在覆盖率和冗余度之间取得平衡的测试序列集合 $\mathcal{P}^*$：

$$\mathcal{P}^* = \arg \max_{\mathcal{P} \subseteq \text{AllPaths}} \text{Coverage}(\bigcup_{P_i \in \mathcal{P}} P_i) - \lambda \cdot \text{Redundancy}(\mathcal{P}) \quad (9)$$

其中 λ 是控制冗余惩罚的权重。最后，对于生成的每条抽象路径，我们将调用一个数据生成模块，为其填充具体的测试数据，使其成为可执行的集成测试用例。

3.2.4 端到端测试执行与覆盖率反馈优化

本步骤旨在构建一个闭环反馈系统，将生成测试序列的自动化执行结果用于指导和优化上游的事件提取、依赖推理和路径规划模型，从而形成一个能够持续自我改进的智能化测试框架。此过程的核心是强化学习。

首先，我们将搭建一个集成了持续集成/持续部署（CI/CD）理念的自动化测试平台。该平台负责接收生成的测试序列 P，自动部署待测系统，执行测试脚本，并监控执行结果。执行结束后，平台会收集丰富的反馈信息，包括：测试序列是否完整执行无异常中断、是否触发了预定义的断言失败（发现缺陷）、以及该序列执行后在代码层面（如函数、分支）和业务层面（如图节点、边）带来的覆盖率增益 ΔCoverage。

其次，我们将这些反馈信号整合成一个综合的奖励函数 $R(P)$，用于评价所生成测试序列 P 的“质量”：

$$R(P) = w_e R_{exec}(P) + w_c \Delta\text{Coverage}(P) + w_b R_{bug}(P) \quad (10)$$

其中，R_{exec} 代表执行成功奖励，R_{bug} 是发现缺陷时的巨大正向奖励，w_e, w_c, w_b 是用于调整各部分重要性的权重。整个测试生成系统被视为一个统一的策略网络 π_θ，其“动作”是生成一个完整的测试序列。我们将采用策略梯度（Policy Gradient）方法，根据获得的奖励信号来更新模型参数 θ。其梯度更新公式为：

$$\nabla_\theta J(\theta) = \mathbb{E}_{P\sim\pi_\theta}[R(P)\nabla_\theta \log \pi_\theta(P|\{R_k\})] \quad (11)$$

此公式的核心思想是“奖优罚劣”：能够带来高奖励（如发现缺陷、提升覆盖率）的生成决策（无论是在事件提取、依赖推理还是路径规划阶段）将被强化，反之则被抑制。此外，为了动态优化奖励函数本身，我们还可以引入贝叶斯优化来调整权重向量 $\mathbf{w} = (w_e, w_c, w_b)$，寻找能最快提升长期缺陷发现率的奖励策略：

$$\mathbf{w}^* = \arg\max_{\mathbf{w}} \int \text{BugRate}(\pi_{\theta(\mathbf{w})}) P(\theta|\mathcal{D}) d\theta \quad (12)$$

通过这个端到端的反馈优化闭环，整个测试生成系统的智能水平将不断提升。