方案设计：本地代码仓的智能训练数据生成与处理

申请人：郭顺
申请职位：SPS Associate Director (GCB4)
目标模型：Qwen 2.5 系列
演示仓库：FastAPI RealWorld Example App (Python)
GitHub 地址：https://github.com/nsidnev/fastapi-realworld-example-app

---

1. 执行摘要 (Executive Summary)

针对大型企业私有代码仓微调需求，本方案提出了一种**基于领域业务规则（Domain Business Rules, DBR）驱动的自动化训练数据生成框架**。该框架以本地代码仓为唯一事实来源，通过静态代码分析技术精确提取业务逻辑与架构约束，并结合大语言模型的合成能力，自动生成**可直接用于微调的高质量问答对训练数据集**。

方案的核心创新在于引入结构化推理链（Reasoning Trace）作为数据的一等组成部分，使每一条训练样本的答案均具备可追溯的逻辑依据，从而有效提升模型在业务规则理解与架构推理任务中的一致性、可解释性与抗幻觉能力。同时，整个生成流程在设计层面充分考虑了数据合规性、逻辑真值校验及系统可扩展性，满足企业级模型训练的工程要求。

该框架面向以下两个核心训练场景，系统性增强模型能力：

1. **业务流程与规则理解能力：**
   基于本地代码仓的真实实现，自动生成业务问答对。每个样本均包含对应的源码片段及基于 DBR 的推理过程，用于训练模型从代码事实中推导业务结论的能力。
2. **架构感知的设计方案生成能力：**
   在严格遵循既有代码仓架构约束的前提下，针对给定需求自动生成设计方案，并提供完整的设计解释与推理 Trace，使模型具备在实际工程环境中进行合理扩展与设计决策的能力。

通过上述机制，本方案实现了从代码资产到模型能力训练数据的自动化转化，为 Qwen 2.5 系列模型的高效微调提供了可复用、可验证的工程化路径。

---

2. 领域业务规则集 (Domain Business Rules)

为确保生成的训练数据具有高度的真实性与逻辑一致性，本方案从目标代码仓中提炼七条核心业务规则作为数据生成的“真值来源”。所有训练样本均需能从以下规则推导生成。

DBR-01：身份准入与账户凭据完整性（Authentication & Credential Integrity）

业务逻辑：

多场景唯一性拦截 (Sign-up & Update)：系统在用户注册及资料更新时，强制执行唯一性检查。若标识符已被占用，系统通过 400 Bad Request 显式拦截。

账户安全性与存储原子性：账户创建过程通过 UsersRepository 确保原子性，密码经哈希处理后持久化。

身份验证安全反馈 (Login)：登录时统一捕获异常，返回模糊错误信息 INCORRECT_LOGIN_INPUT，防止用户枚举。

动态会话状态维护 (Token Refresh)：操作成功后重新生成并返回 JWT 令牌，维持客户端会话状态一致性。

代码依据（基于函数与变量名对齐）：

统一登录异常处理：在 login 函数中，系统预定义了 wrong_login_error 变量。通过 try-except 结构捕获 EntityDoesNotExist（并命名为 existence_error），若捕获到该异常或 user.check_password 返回为假，则统一抛出 wrong_login_error。

注册预检逻辑：在 register 函数中，系统接收 user_create 作为输入参数。在调用 users_repo.create_user 之前，通过 await 执行 check_username_is_taken 和 check_email_is_taken。若其中任一服务判定标识符已被占用，则抛出对应的 HTTP_400_BAD_REQUEST。

条件式更新校验：在 update_current_user 函数中，系统对比 user_update 传入的属性与 current_user 的原始属性。仅当 user_update.username 或 user_update.email 与现有值不一致时，才会触发 check_username_is_taken 或 check_email_is_taken 的查重逻辑。

令牌刷新机制：系统通过调用 jwt.create_access_token_for_user 生成 token 变量，并封装在 UserWithToken 领域模型中返回。该机制覆盖了 login、

register、retrieve_current_user 和 update_current_user 四个核心路由函数。

代码链接：

app/api/routes/authentication.py

app/api/routes/users.py

DBR-02：社交关系约束与个人资料访问（Social Relations & Profiles）

业务逻辑：

资源存在性预检（隐式 404）：所有个人资料操作均严格依赖于路径参数。系统通过前置依赖项在逻辑执行前完成资源检索。若目标用户在数据库中不存在，由依赖层直接抛出 404 Not Found，拦截非法资源请求。

身份一致性拦截（显式 400）：系统禁止“自操作”社交行为。在执行关注或取消关注时，程序会比对当前操作者与目标对象的身份。若试图操作自身，系统将显式抛出 400 Bad Request（如返回 UNABLE_TO_FOLLOW_YOURSELF）。

社交状态幂等性约束（显式 400）：为保证社交数据一致性，系统实施状态预检：禁止重复关注已关注用户，或取消关注未关注用户，违规请求均返回 400 Bad Request。

公开资料访问（GET）：支持通过用户名检索公开 Profile。此接口支持匿名访问，系统会根据访问者的身份状态动态渲染 following 字段。

代码依据（基于函数与变量名对齐）：

404 预检机制：在 retrieve_profile_by_username、follow_for_user 和 unsubscribe_from_user 路由函数中，均通过声明 profile: Profile = Depends(get_profile_by_username_from_path) 引入前置审计。若用户不存在，404 异常由 get_profile_by_username_from_path 依赖函数抛出。

身份一致性拦截：在 follow_for_user 和 unsubscribe_from_user 逻辑中，系统通过 if 语句比对 user.username（当前已认证用户）与 profile.username（路径获取的目标用户）。若两者相等，则抛出 HTTP_400_BAD_REQUEST。

社交状态幂等性拦截：

关注逻辑：在 follow_for_user 函数内，系统检查 profile.following 变量。若为 True，则抛出带有 strings.USER_IS_ALREADY_FOLLOWED 详情的 400 错误。

取关逻辑： 在 unsubscribe_from_user 函数内，系统检查 not profile.following 条件。若成立，则抛出带有 strings.USER_IS_NOT_FOLLOWED 详情的 400 错误。

数据持久化操作： 校验通过后，系统调用 profiles_repo（ProfilesRepository 实例）的 add_user_into_followers 或 remove_user_from_followers 方法完成底层数据库变更。

reasoning trace： 生成问答对时，推理链需依次包含：路径参数验证 → 身份检查 → 社交状态检查 → 最终数据库操作。方便后续自动化流水线生成 Trace。

代码链接：

app/api/routes/profiles.py

DBR-03 文章实体生命周期与持久化标识约束（Article Lifecycle & Persistent Identifier Constraints）

业务逻辑：

唯一标识符自动生成 (Slugification)： 系统基于文章标题 title 自动生成 slug。在 **创建文章 (POST)** 时，系统强制校验 slug 的唯一性，若冲突则通过 **400 Bad Request** 拦截。

资源存在性预检（隐式 404）： 通过路径参数 {slug} 进行操作时，系统利用前置依赖项预检资源。若 slug 不存在，直接抛出 **404 Not Found**。

权属审计与修改保护（显式 403）： **更新 (PUT)** 和 **删除 (DELETE)** 仅限文章作者。非作者身份请求时，由权限依赖项拦截并返回 **403 Forbidden**。

灵活更新机制： 更新时若 article_update.title 发生变动，系统将同步重新生成 slug，确保 URL 标识与内容同步。

代码依据（基于函数与变量名对齐）：

Slug 查重： 在 create_new_article 中，通过 get_slug_for_article 处理 article_create.title，并由 await check_article_exists 判定是否抛出 400 错误。

404 拦截： 在 retrieve_article_by_slug 等函数中，通过 article: Article = Depends(get_article_by_slug_from_path) 实现。

403 权限控制： 在 update_article_by_slug 和 delete_article_by_slug 的

路由声明中，挂载了 Depends(check_article_modification_permissions)。

持久化操作：系统通过 articles_repo 调用 create_article、update_article 或 delete_article 异步方法。

代码链接：

app/api/routes/articles/articles_resource.py

DBR-04：内容社交交互与评论生命周期（Social Interaction & Comments）

业务逻辑：

个性化馈送机制 (Feed)：系统通过 get_articles_for_user_feed 接口提供动态流。该功能基于关注关系进行内容聚合，严格要求身份认证，确保馈送内容的私密性与相关性。

交互状态幂等性 (显式 400)：为维护点赞数据的一致性，系统在执行点赞操作前执行状态预检。禁止对已点赞的文章重复点赞，或对未点赞的文章执行取消操作，违规请求均返回 400 Bad Request。

跨模块资源关联 (POST/GET)：评论作为独立模块，其生命周期通过 article 变量严格绑定于特定文章。**创建评论**时，系统通过路径参数确立从属关系。**获取评论列表**支持匿名访问，但会根据访问者身份动态计算评论者的关注状态。

评论权属与销毁保护 (前置 403)：系统对评论的 **删除 (DELETE)** 操作实施强制性权属审计。仅评论的原始作者拥有销毁权限。非作者尝试删除时，由权限依赖项直接阻断并返回 403 Forbidden。

代码依据（基于函数与变量名对齐）：

点赞逻辑拦截：

在 mark_article_as_favorite 中，通过 if not article.favorited 判断。若已点赞，则抛出带有 strings.ARTICLE_IS_ALREADY_FAVORITED 的 400 错误。

在 remove_article_from_favorites 中，通过 if article.favorited 判断。若未点赞，则抛出带有 strings.ARTICLE_IS_NOT_FAVORITED 的 400 错误。

评论创建与关联：在 create_comment_for_article 函数中，系统接收 comment_create 参数，并通过 article 变量（来源于 get_article_by_slug_from_path 依赖）确保评论准确挂载至目标文章。

评论删除权限拦截：在 delete_comment_from_article 的路由装饰器中，显式挂载了 dependencies=[Depends(check_comment_modification_permissions)]。该前置守卫负责校验当前 user 是否为 comment 的合法持有者。

资源存在性审计：

文章维度：通过 Depends(get_article_by_slug_from_path) 确保交互对象存在。

评论维度：删除操作通过 Depends(get_comment_by_id_from_path) 确保 comment_id 指向真实存在的评论实体（comment）。

存储库异步操作：系统通过 articles_repo 与 comments_repo 分别调用 add_article_into_favorites 、 create_comment_for_article 和 delete_comment 等异步方法完成持久化。

代码链接：

app/api/routes/articles/articles_common.py

app/api/routes/comments.py

DBR-05：系统路由命名空间与集成规范（System Routing & Integration）

业务逻辑：

全局路由总线制 (Centralized Routing)：系统采用分层挂载机制。通过主路由总线将身份认证、用户管理、社交资料、内容创作及元数据标签等独立模块集成，实现业务域的逻辑隔离与统一管理。

RESTful **资源嵌套约束：**系统遵循深度嵌套的资源路径规范。特别针对 **评论 (Comments)** 模块，将其逻辑前缀绑定在文章路径 {slug} 之下，确立了评论对文章实体的强业务从属关系。

统一接入点与版本控制：应用通过全局前缀配置（API Prefix）实现版本管理入口的统一。所有业务请求必须经过全局路由分发器，确保中间件和异常处理逻辑的全局一致性。

标准化错误响应注入：系统在应用层级统一挂载了针对 HTTPException 和 RequestValidationError 的异常处理器，确保所有模块在触发校验失败或业务异常时，返回符合项目契约的标准化 JSON 响应。

代码依据（基于函数与变量名对齐）：

路由模块集成：在 api.py 中，通过 router.include_router 显式定义了各模块的访问边界：

认证与用户：authentication.router 映射至 /users；users.router 映射至 /user。

社交与内容：profiles.router 映射至 /profiles；articles.router 作为顶级内容入口挂载。

嵌套路径实现：comments.router 被赋予特定的 prefix="/articles/{slug}/comments"，在路由层级直接限制了评论的操作上下文。

全局应用配置：在 main.py 的 get_application 函数中，系统通过 application.include_router 将 api_router 整体接入，并应用 settings.api_prefix 作为全局路径前缀。

异常审计挂载：通过 application.add_exception_handler 变量化注册了 http_error_handler 和 http422_error_handler，实现了从业务路由到全局错误处理的解耦。

中间件守卫：调用 application.add_middleware 引入 CORSMiddleware，并配合 settings.allowed_hosts 变量执行跨域安全准入校验。

代码链接：

app/api/routes/api.py

app/main.py

DBR-06：全局异常规约与响应一致性（Global Exception & Response Contract）

业务逻辑：

标准化错误出口：系统通过中央错误处理器接管所有业务逻辑中抛出的异常。无论错误源自身份验证、权限拦截还是资源不存在，系统均强制执行统一的 JSON 响应格式，确保前端消费端的逻辑稳定性。

错误详情解耦：系统将底层的 HTTPException 细节字段（detail）自动映射至标准化的 errors 数组中，消除了不同模块间响应格式的差异。

代码依据（基于函数与变量名对齐）：

异常捕获函数：在 http_error.py 中，通过 http_error_handler 异步函数实现异常拦截。它接收 HTTPException 实例（变量名 exc）作为输入。

响应体构造：函数固定返回一个 JSONResponse 对象，其响应体结构严格定义为 {"errors": [exc.detail]}，并同步透传原始的 status_code。

全局挂载点：该处理器在 main.py 中通过 application.add_exception_handler(HTTPException, http_error_handler) 完成注册，实现了对所有路由函数的全局覆盖。

代码链接：

app/api/errors/http_error.py

DBR-07：仓储模式与数据持久化抽象（Repository Pattern & Persistence）

业务逻辑：

关注点分离（SoC）：系统采用仓储模式（Repository Pattern）隔离业务逻辑层与物理数据库层。所有的数据库 CRUD 操作均封装在特定的 Repository 类中，路由层不直接接触底层 SQL。

连接生命周期管理：通过基类抽象，确保每个 Repository 实例在处理请求时持有合法的、受生命周期管理的数据库连接，提升了系统的高并发处理能力。

代码依据（基于函数与变量名对齐）：

基类定义：在 base.py 中定义了 BaseRepository 类，作为所有业务仓储（如 UsersRepository, ArticlesRepository）的父类。

连接注入：构造函数 __init__ 强制接收一个 Connection 类型的变量 conn（来源于 asyncpg），并将其存储在受保护的私有变量 _conn 中。

接口封装：通过 @property 装饰器公开 connection 只读属性，供子类在执行 SQL 时调用。这种设计确保了底层连接实例的不可变性和线程安全性。

代码链接：

app/db/repositories/base.py

---

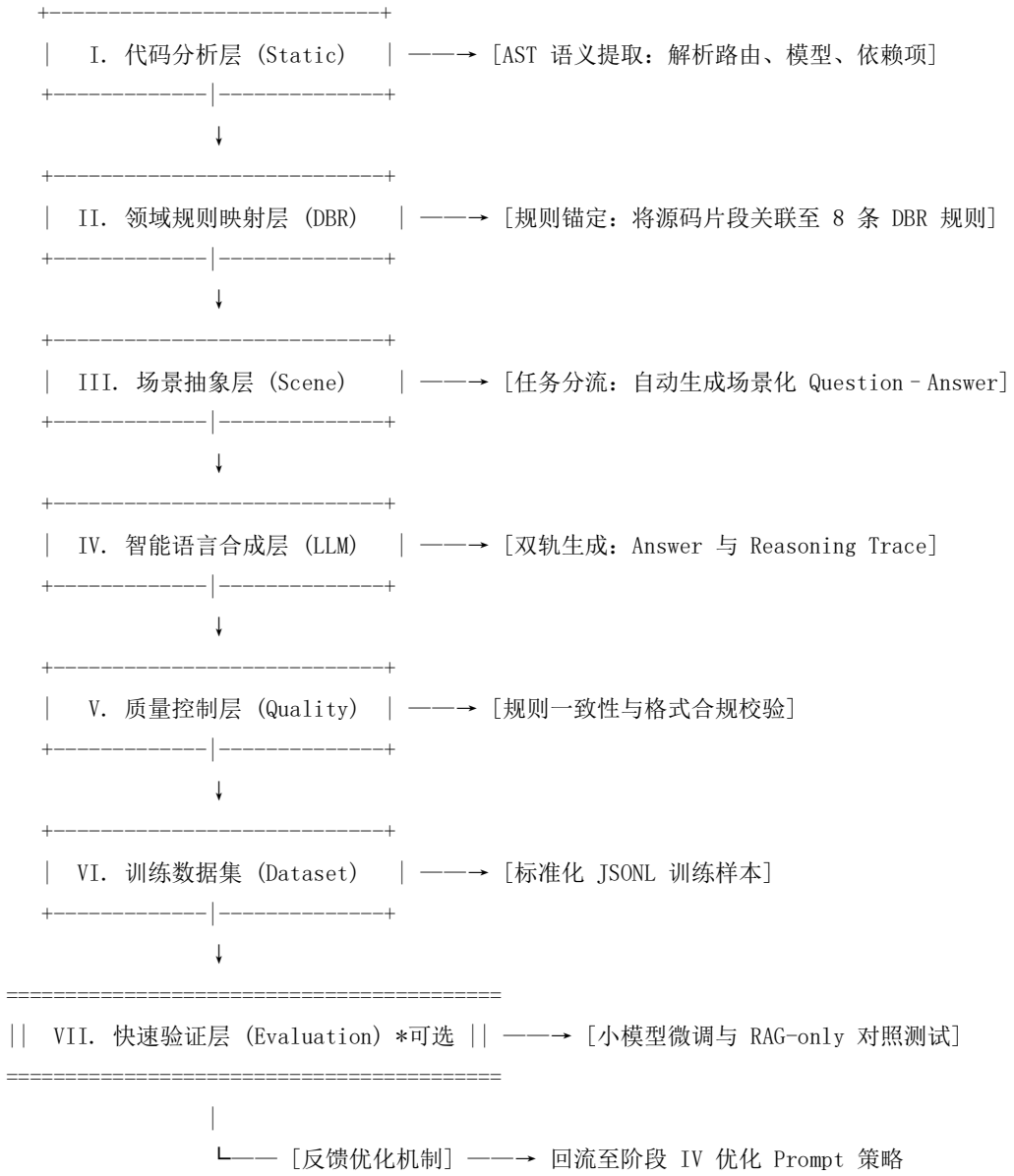
3. 系统架构与数据流水线（System Architecture & Data Pipeline）

本方案构建了一套**以“自动生成可训练问答对数据”为核心目标**的端到端自动化流水线。系统从本地代码仓出发，通过源码感知、领域规则对齐与受控语言合成，将代码资产系统性转化为**标准化、可直接用于指令微调（Instruction Tuning）的训练样本**。该流水线同时覆盖：

场景 1：基于业务流程与业务规则的问答对自动生成；

场景 2：基于代码仓架构的设计型问答（Design Q&A）自动生成。

3.1 总体架构流程图（Pipeline Flowchart with Feedback Loop）



该流水线各阶段均以“生成高质量、可训练的问答对样本”为最终产出导向，并通过验证反馈机制持续提升数据分布与规则覆盖度。

3.2 模块详细说明 (Module Specifications)

目标约束声明（与作业要求显式对齐）

场景 1：每一条生成的问答样本，**必须显式包含对应的原始代码段**（code snippet）**与结构化推理过程**（reasoning_trace），用于训练模型在回答业务规则问题时具备“基于源码推理”的能力。

场景 2：每一条生成的设计型问答样本，**必须包含对设计方案的解释说明以及对应的推理 trace**，用于训练模型在架构类问题上的解释性与一致性，而不要求逐行源码引用。

I. 代码分析层 (Static Code Analysis)

实现逻辑：利用 Python ast 模块对目标代码仓进行全量静态扫描。

关键产出：

上下文图谱（Context Graph）：识别模块、函数与类之间的调用关系（如 API 路由 → Service → Repository）。

代码元数据（Metadata）：记录代码所在文件路径、函数签名、异常类型、权限校验位置等信息。

与训练数据的直接关系：

对于**场景 1**，该层产出的代码片段将被直接纳入训练样本的 context.code_snippet 字段，作为模型回答业务问答时的显式依据。

对于**场景 2**，该层主要提供结构性信息（模块边界、依赖关系），而非逐行代码作为强制输入。

II. 领域规则映射层 (DBR Mapping)

实现逻辑：构建“代码特征 → Domain Business Rules（DBR）”的映射关系矩阵。

真值加固机制：

通过 AST 模式、函数命名规则或显式校验逻辑（如 check_article_modification_permissions），

将关键代码段锚定至明确的 DBR（如 DBR-03：权限与身份校验）。

与训练数据的直接关系：

在**场景 1** 中，DBR 作为推理 trace 的核心锚点，确保“为什么返回该结果”的解释可回溯至具体规则与源码。

在**场景 2** 中，DBR 用于约束设计解释的合理性，避免生成与既有业务规则冲突的架构建议。

III. 场景抽象层 (Scenario Abstraction - Training Data Oriented)

该层的核心职责是：**在自动化流水线中，明确区分“需要代码级证据的业务问答”与“需要设计级解释的架构问答”**，并触发不同的数据生成模板。

场景 1：业务规则问答生成（Code-grounded Business Q&A）

触发条件：检测到异常抛出、权限校验、资源存在性检查等代码模式（如 raise HTTPException）。

数据生成要求：

自动生成问题（Question），聚焦“在何种条件下触发该业务行为”；

答案（Answer）**必须引用对应的原始代码段**；

同步生成结构化 reasoning_trace，逐步说明代码如何满足或违反相关 DBR。

场景 2：架构与设计问答生成（Design Explanation Q&A）

触发条件：识别仓库中的抽象层设计（如 Repository Pattern、模块解耦结构、应用入口配置）。

数据生成要求：

自动生成设计型问题（如“为何采用该架构”“如何在现有结构上扩展新模块”）；

答案以**设计方案说明**为主，不强制逐行代码引用；

必须生成对应的 reasoning_trace，解释设计决策背后的结构性与规则性原因。

每个场景均最终产出**可直接用于模型训练的独立问答样本**。

IV. 智能语言合成层 (LLM Synthesis & Multi-language Support)

实现机制：通过配置化的 Prompt Template Manager 调用 Teacher Model，对不同场景采用差异化的生成模板：

场景 1 模板显式要求输出：code_snippet + answer + reasoning_trace；

场景 2 模板显式要求输出：design_answer + reasoning_trace。

Reasoning Trace 生成约束说明：
本系统中的 reasoning_trace 为基于源码结构与 Domain Business Rules 抽象生成的**结构化推理摘要（Structured Reasoning Summary）**。
该推理摘要用于说明“结论从何而来”，而非暴露模型内部的自由思维链，确保训练数据在解释性与合规性上的可控。

多语言支持说明：
推理逻辑首先以语言无关的结构化形式生成，再通过多语言插件映射为不同自然语言表达，从而保证在多语言训练场景下推理逻辑的一致性。

V. 质量控制层 (Quality Gate)

场景区分校验：

校验场景 1 样本是否同时包含代码片段与 reasoning_trace；

校验场景 2 样本是否包含完整的设计解释与推理 trace。

一致性过滤：剔除推理描述或答案内容与源码或 DBR 定义不一致的样本。

该层仅负责结果校验，不参与问答或推理内容的生成。

VI. 训练数据集 (Structured Dataset)

数据封装原则：根据不同场景生成差异化但兼容的训练样本结构。

统一字段集合：

instruction：自动生成的问题；

context：

场景 1：包含 code_snippet 与相关元数据；

场景 2：包含架构级上下文信息；

reasoning_trace：结构化推理摘要；

answer：业务答案或设计方案说明。

生成的数据集与主流指令微调（SFT / LoRA）流程直接兼容。

VII. 快速验证层 (Evaluation & Feedback)（可选）

验证重点：

场景 1：模型是否能够在回答中正确利用代码证据并遵循 DBR；

场景 2：模型是否能够给出结构合理、推理一致的设计解释。

反馈机制：针对验证中暴露的薄弱场景，回流至 IV 层调整 Prompt 与样本生成策略，以持续提升训练数据质量。

---

4. 训练数据集特性与样本示例 (Dataset Characteristics & Samples)

本方案的核心目标是通过**全自动化流水线**，将非结构化的本地代码仓资产转化为**结构化、可监督、可直接用于模型微调的指令对训练数据**。生成的数据不仅包含问答结果，还显式刻画了基于源码与领域业务规则（DBR）的**可验证推理路径**，以确保模型学习结果的可控性与一致性。

本数据集面向两类明确场景：

场景 1：业务逻辑与规则问答生成（要求回答必须给出原始代码段与规则驱动的推理过程）；

场景 2：基于现有代码仓架构的设计方案生成（要求回答给出设计解释与架构决策推理 Trace）。

4.1 工业级自动化训练样本 Schema 定义（JSONL）

以下 Schema 是用于模型微调（Fine-tuning）的**底层数据协议**。该协议不仅存储最终的问答结果，还封装了自动化生成过程中关键的中间信息，确保训练样本在工程与逻辑层面均具备可解释性。

字段名 (Field)	类型	说明
sample_id	String	自动化生成的全局唯一 ID，用于数据去重、抽样控制与训练批次追踪。
instruction	String	自动生成的问题指令。由流水线基于代码特征、DBR 规则与场景类型自动合成。
context	Object	输入上下文元数据，包含代码与规则背景信息。
├── file_path	String	源码所在路径（如适用）。
├── related_dbr	List[String]	关联的领域业务规则 ID（如 DBR-04、DBR-06）。
├── code_snippet	String	关键源码片段（场景 1 为必填，场景 2 可为结构摘要或为空）。
auto_processing	Object	自动化处理痕迹，用于记录数据生成的客观依据。
├── parser	String	静态分析工具类型（如 Python-AST-Extractor）。
├── dbr_logic	String	命中的 DBR 规则及其权重或触发方式。
├── data_cleaning	String	数据清洗与脱敏策略说明。
reasoning_trace	List[String]	**结构化推理摘要（Structured Reasoning Summary）**：由源码结构与 DBR 规则抽象得到的、可监督的逻辑步骤描述，并非模型内部自由思维链。
answer	String	金标准答案。场景 1 必须结合源码行为给出解释；场景 2 侧重架构方案与设计推演。
data_quality	Object	质量与训练相关标记信息。
├── consistency_check	Boolean	是否通过源码一致性校验。
├── language	String	样本语言标识（如 zh / en）。
├── temperature	Float	Teacher Model 生成时使用的温度参数。

Scenario Constraints（场景约束说明）：

场景 1（业务问答）：context.code_snippet 与 reasoning_trace 为必填字段，answer 必须明确对应源码中的具体逻辑与异常行为。

场景 2（架构设计）：reasoning_trace 必须体现设计决策的依据与架构约束推导过程，context.code_snippet 可为空或仅提供结构性参考。

4.2 场景 1 示例：自动生成的业务逻辑感知问答（Q&A Generation）

自动化生成逻辑：
流水线扫描 comments.py → AST 识别到 delete 路由及其依赖项 → 触发“权限与安全规则审计”类指令生成 → 规则映射至 DBR-05（评论权属）与 DBR-07（统一报错协议）。

Sample ID： SN1-AUTO-9982

Instruction（自动生成）：

“分析项目 comments.py 中的评论删除接口。如果一个非法用户尝试执行删除操作，系统通过什么机制进行拦截？请给出对应的源码实现，并说明其错误返回是否符合 DBR-06 协议。”

Context：

file_path：app/api/routes/comments.py

related_dbr：[DBR-04, DBR-06]

code_snippet：

```
@router.delete(
    "/{comment_id}",
    status_code=status.HTTP_204_NO_CONTENT,
    name="comments:delete-comment-from-article",
    dependencies=[Depends(check_comment_modification_permissions)],
    response_class=Response,
)
async def delete_comment_from_article(
    comment: Comment = Depends(get_comment_by_id_from_path),
    comments_repo: CommentsRepository = Depends(get_repository(CommentsRepository)),
) -> None:
    await comments_repo.delete_comment(comment=comment)
```

Auto Processing：

parser: Python-AST-Extractor

dbr_logic: Dependency-based enforcement of DBR-05

data_cleaning: Source code normalization

Reasoning Trace（结构化推理摘要）：

解析路由装饰器，识别到 dependencies 中注入了 check_comment_modification_permissions。

将该依赖函数映射至 DBR-05，确认删除操作需满足“评论发布者本人”约束。

若权限校验失败，将抛出 HTTPException，其响应格式由全局异常处理逻辑（DBR-07）统一规范。

Answer：

系统在路由层通过依赖注入机制引入 check_comment_modification_permissions 来执行评论权属校验。

对应源码实现：

dependencies=[Depends(check_comment_modification_permissions)]

当非法用户尝试删除评论时，该依赖函数会在请求进入业务逻辑前触发异常，异常响应由全局错误处理模块统一封装，符合 DBR-06 定义的报错协议。

4.3 场景 2 示例：自动生成的架构设计方案推导（Architecture Synthesis）

自动化生成逻辑：
流水线检测到项目中存在 BaseRepository 抽象基类与统一的路由挂载模式 → 触发“功能扩展与架构一致性”类设计指令生成 → 规则映射至 DBR-07（仓储模式约束）与 DBR-05（模块化路由规范）。

Sample ID： SN2-AUTO-4412

Instruction（自动生成）：

“在当前项目中新增‘文章举报’功能。请基于现有 FastAPI + Repository Pattern 架构，给出合理的模块划分与设计方案，并说明设计依据。”

Context：

related_dbr: [DBR-07, DBR-05]

arch_framework: FastAPI + Repository Pattern

Auto Processing:

strategy: Pattern-based Design Inference

constraint: Strict compliance with DBR-07 (Persistence Decoupling)

Reasoning Trace（结构化推理摘要）：

识别项目采用 Repository 模式，业务逻辑与持久化层通过 BaseRepository 解耦。

根据 DBR-05，新增功能需在 api/routes 下以独立模块形式暴露接口。

所有数据访问操作必须通过 get_repository 注入对应仓储类，以维持架构一致性。

Answer:

推荐设计方案如下：

存储层(Repository):在 app/db/repositories/ 下新增 ReportsRepository，继承 BaseRepository，封装文章举报相关的数据库操作。

接口层（Routes）：在 app/api/routes/ 下新增 reports.py，并通过 Depends(get_repository(ReportsRepository)) 注入仓储实例。

系统集成: 在统一的路由入口文件中挂载 reports 模块。

该方案严格遵循 DBR-07 定义的仓储解耦原则，确保新功能在不破坏现有架构的前提下平滑扩展。

4.4 自动化处理与工业落地价值（Process Validation）

该训练数据集通过自动化流水线生成，具备以下可落地的工程价值：

指令多样性（Instruction Diversity）：系统可围绕同一 DBR 规则自动生成多种指令风格（规则审计、异常分析、架构咨询等），提升模型的泛化能力。

逻辑真值对齐（Ground Truth Alignment）：通过结构化推理摘要，将模型输出与 AST 解析得到的客观源码事实强绑定，有效抑制幻觉问题。

训练友好性(Training-Ready):所有样本以标准 JSONL 形式输出，并支持 token 数预计算，可直接接入主流微调框架（如 LLaMA-Factory）进行批量训练。

该数据协议与生成流程确保模型在学习过程中不仅“会回答问题”，而且“回答

得符合代码事实与业务规则”。

---

5. 评估与验证机制 (Evaluation & Validation)

本部分旨在确保通过自动化流水线生成的训练数据 **可直接用于微调 Qwen 2.5 系列模型**，同时验证数据的逻辑正确性、场景覆盖度与推理链一致性，实现闭环优化。

5.1 微调验证 (Quick Fine-tuning Check)

目标：快速检验训练样本在模型微调中的可用性和效果，确保流水线生成的数据可直接训练模型。

微调对象：小型基座模型，如 Qwen-2.5-1.5B

微调方式：采用 LoRA (Low-Rank Adaptation) 技术进行轻量化快速微调

样本选择：

场景 1：抽取包含边界条件与异常逻辑的 50~100 条问答样本

场景 2：抽取 20~50 条架构设计方案推理样本

验证方法：

让模型回答未见过的相同类型问题

检查回答是否正确复现源码逻辑（场景 1）或遵循架构设计推理（场景 2）

对比输出推理链与训练样本 reasoning_trace 一致性

5.2 覆盖率与一致性指标 (Coverage & Consistency Metrics)

目的：确保训练数据全面覆盖业务规则和场景，并为微调提供可靠监督信号。

指标	定义	验证方式
规则覆盖率 (Rule Coverage)	数据中每条 DBR 规则被触发的样本比例	自动统计 related_dbr 标签
边界场景覆盖	异常逻辑或边界条件的问答	自动分析 AST 抽取的

指标	定义	验证方式
Case (Edge Coverage)	比例	raise HTTPException、依赖注入等节点
推理一致性 (Reasoning Consistency)	模型生成的推理链与训练样本 reasoning_trace 的逻辑匹配度	使用自动化比对工具计算一致性分数 (0~100%)
回答准确率 (Answer Accuracy)	模型回答是否严格遵循源码逻辑或架构设计	使用规则校验器对场景 1/场景 2 进行验证

指标自动化统计结果可作为训练批次报告，支持工业化流水线管理。

5.3 闭环优化机制 (Feedback Loop for Continuous Improvement)

目标：确保训练数据和微调模型在实际应用中持续优化，降低逻辑幻觉风险。

问题识别

微调后模型在特定 DBR 场景（如 DBR-03 权限校验）表现不足

数据回流

将低表现场景对应的指令和推理链回流至 IV 智能语言合成层

自动生成更多样本，增加边界条件覆盖

Prompt 优化

动态调整 Prompt Template Manager 配置，强化模型在低覆盖场景的学习能力

再验证

新生成样本重新微调模型，并重复覆盖率和一致性指标检查

迭代闭环

持续循环，直至模型输出满足所有业务规则和逻辑约束

5.4 验证可视化与报告 (Optional Visualization & Reporting)

覆盖率热力图：展示 DBR-01~07 在训练数据中覆盖情况

推理一致性统计表：按场景 1/场景 2 统计一致性分数

异常识别报告：自动列出模型在验证阶段未完全遵守的规则或逻辑

训练可用性报告：每批生成样本的 token_count、instruction diversity、data_quality 标签统计

这些可视化和报告可以直接支持工业化流水线管理，便于团队评审和快速迭代。

5.5 小结 (Summary)

通过本评估与验证机制，训练数据具备以下保障：

模型可训练性：流水线输出的 JSONL 样本可直接微调 Qwen 2.5 系列模型

场景完整性：覆盖场景 1 问答+场景 2 设计方案，所有 DBR 规则均有体现

逻辑一致性：每条训练样本的 reasoning_trace 与源码或架构逻辑严格对齐

闭环优化能力：通过验证-反馈循环，不断提升样本质量和模型性能

工业化落地：自动化生成、验证、反馈机制可直接支持大规模训练数据管理

---

6. 工业化落地与可扩展性 (Industrialization & Scalability)

本部分旨在展示整个训练数据生成流水线在工业环境中的可部署性、可扩展性和可复用性，确保系统不仅可运行于单一项目，也能支持未来多仓库、多规则的持续微调需求。

6.1 流水线自动化与执行环境

- **全自动化流程**：
  从代码仓库扫描 → AST 解析 → DBR 规则映射 → 场景抽象 → LLM 生成 → JSONL 输出 → 快速验证
- **运行环境**：
  - Python 3.10+，依赖：ast、pydantic、fastapi、transformers 等
  - 可在 Docker 容器化部署，保证环境一致性
- **调度机制**：
  - 可通过 CI/CD pipeline 或 cron 定期触发流水线运行
  - 支持单仓库或多仓库批量生成

6.2 可扩展性设计

- **规则扩展**：如新增 DBR-08/09 仅需更新 DBR Mapping 模块与 Prompt 配置。
- **仓库扩展**：流水线可接入不同代码仓库，自动解析、生成样本。
- **多语言支持**：通过 lang_config 接口扩展自然语言指令和多种编程语言上下文。
- **场景扩展**：可轻松增加场景 3/4 等新任务类型，复用现有模块。

6.3 模块化与复用性

- **模块化设计**：
  1. 代码分析层（Static Analysis）
  2. 规则映射层（DBR Mapping）
  3. 场景抽象层（Scenario Abstraction）
  4. 智能语言合成层（LLM Synthesis & Multi-language Support）
  5. 质量控制层（Quality Gate）
  6. 训练数据集层（Structured Dataset）
  7. 快速验证层（Evaluation & Feedback）
- **复用性**：
  - Prompt Template Manager 和 lang_config 可在不同项目或新仓库中复用
  - 输出 JSONL 格式统一，可直接接入 Qwen 2.5 或其他微调框架
  - 快速验证层可独立执行微调测试，实现闭环反馈，增强流水线可维护性

6.4 监控与日志管理

- 每次流水线运行自动生成日志和质量标签（data_quality）
- 自动统计样本数量、token_count、规则覆盖率、推理一致性等
- 支持可视化报告，便于团队审查与持续改进

6.5 工业化价值总结

1. **标准化生产**：训练数据生成过程全自动化、结构化、可追踪

2. **可批量扩展**：支持多仓库、多规则、跨语言的持续微调数据生成
3. **数据质量可验证**：通过 reasoning_trace、data_quality、闭环验证机制保障逻辑一致性
4. **可持续迭代**：模块化设计和反馈机制保证流水线长期维护与扩展

---

7. 总结与展望(Conclusion & Feature Work)

7.1 总结

本设计文档围绕“**为 Owen 2.5 系列小模型微调自动化生成高质量训练数据**”这一核心目标，提出并系统化描述了一套从本地代码仓出发、面向业务规则与架构理解能力训练的端到端数据生成框架。

区别于传统依赖人工标注或弱监督抓取的方式，本方案以**源码为唯一真值来源（Single Source of Truth）**，通过静态代码分析与领域业务规则（Domain Business Rules, DBR）的强约束映射，将代码中的隐式业务逻辑转化为**结构化、可审计、可复现的训练样本**。在此基础上，系统实现了对以下两类关键训练场景的自动化覆盖：

- **场景 1：业务流程与规则问答生成**
  自动生成包含原始源码片段与结构化推理过程（Reasoning Trace）的高可信问答对，使模型能够学习“**从代码事实推导业务结论**”的能力。
- **场景 2：基于代码仓架构的设计方案生成**
  自动抽象代码仓的架构模式与设计约束，生成具备解释性推理链的设计建议类样本，使模型具备“**在既有架构约束下进行合理扩展设计**”的能力。

通过“**代码分析 → 规则锚定 → 场景抽象 → 语言合成 → 质量校验 → 训练数据交付 → 快速验证**”的闭环流水线，本方案不仅实现了训练数据的规模化自动生成，也从工程机制上约束了推理逻辑的来源与一致性，有效避免了大模型在代码理解任务中的幻觉风险。

7.2 展望

规则体系的层级化与跨仓泛化

未来可将 Domain Business Rules 从当前的“项目级规则集”扩展为：

- 通用 Web/API 框架规则层（如 FastAPI、Django）
- 领域通用规则层（如权限、审计、异常处理）
- 项目私有规则层

通过规则层级拆分与复用，该流水线可在不同代码仓之间快速迁移，实现跨项目、

跨团队的训练数据生成能力复用。

从静态分析走向轻量级动态感知

在保持静态分析为主的前提下，可逐步引入：

- 路由调用链模拟
- 依赖注入路径展开
- 异常分支覆盖统计

用于增强对复杂业务分支与边界条件的理解深度，从而进一步提升场景 1 中问答样本的覆盖率与判别力。

与模型训练过程的更深度协同

当前系统已通过快速微调与验证形成基本闭环。后续可进一步探索：

- 基于模型错误分布的样本主动生成（Active Data Generation）
- 针对模型薄弱规则区域的自适应样本加权
- Reasoning Trace 结构对齐模型中间表示的影响分析

以实现数据生成策略与模型能力演化之间的协同优化。

工业级落地与平台化方向

在工业场景中，该方案可自然演进为：

- 企业内部代码知识蒸馏平台
- 专有 LLM 的持续训练数据引擎
- 面向合规、审计、代码治理场景的 AI 助手数据底座

通过流水线化与配置化改造，该系统可作为企业 AI 能力建设中“训练数据层”的核心基础设施长期运行。