Task1

1. 是否需要设计数据库?

- **MySQL 的作用：** 存储元数据（文本内容、类目树结构、生效时间、视角、标签、关联问题 ID 等）。
  - **Category Table (类目表):** 存储 `id`, `parent_id`, `name`, `level` 等，用于实现无限级分类。
  - **FAQ Main Table (主表):** 存储标准问、答案、答案类型、有效期、视角等。
  - **Similar Questions Table (相似问表):** 一对多关联主表，存储那 200 条相似问。
  - **Tag/Perspective Mapping (关联表):** 存储标签和视角的关联关系。
- **Milvus 的作用：** 仅存储 `Primary Key`（对应 MySQL 的 ID）和 `Vector`（向量特征）。

2. 需要使用什么模型?

- **Embedding Model:** 推荐使用 **BGE-M3** 或 **BERT-based** 模型。
- **Rerank Model:** 在 Milvus 返回 Top 10 后，再用 `BGE-Reranker` 进行精排，选出得分最高的一个。

3. 如何使用 BERT?

在 FAQ 场景中，BERT 通常不是用来"生成"回答，而是用来**向量化（Encoding）**。

1. **预处理：** 将标准问和相似问进行分词和清洗。
2. **向量提取：** 将文本输入预训练好的 BERT 模型，取 `[CLS]` 位的输出或 `Mean Pooling` 层的输出作为该问题的 **768 维特征向量**。
3. **入库：** 将向量存入 Milvus。
4. **检索：** 用户提问时，同样通过 BERT 转化为向量，在 Milvus 中计算**余弦相似度（Cosine Similarity）**。

4. 是否需要使用大模型（LLM）？

- **如果不使用 LLM：** 这是一个标准的 **语义匹配 FAQ**。用户问什么，你从数据库原封不动拿答案吐出来。优点是回复受控、准确，不会"胡言乱语"。
- **如果使用 LLM：** 它可以作为 **"润色"或"理解"层**：
  - **纠错/扩充：** 用户提问口语化严重时，LLM 先将其转化为标准表达。
  - **生成式回答：** 如果 FAQ 没直接命中，LLM 可以检索相关的几个 FAQ 答案，融合成一段通顺的话回复用户（即 RAG 模式）。
  - **意图识别：** 判断用户是在咨询、闲聊还是投诉。

5. 后端与算法分工

角色	核心任务
后端开发	1. 设计 MySQL 表结构，实现类目树的 CRUD。 2. 封装 Milvus 的操作 SDK（增删改查向量）。 3. 逻辑控制：如生效时间过滤、视角筛选。
算法开发	1. 选型并部署 Embedding 模型服务（建议用 FastAPI 或 Triton）。 2. 优化向量检索精度（如处理长短句匹配、停用词）。 3. 建立离线/增量索引更新机制。

Task2

1. **分词 (Tokenization)：** 使用 BERT 的 `Tokenizer` 将输入的 FAQ 标题或相似问切分为 Token，并添加特殊占位符 `[CLS]` 和 `[SEP]`。
2. **特征提取：** 将输入喂入 BERT 编码器，得到每一层 Token 的 Embedding并进行池化。
3. **归一化 (Normalization)：** 对输出向量进行 $L2$ 归一化。

在生产环境中，相似度计算分为两个阶段：**离线索引**和**在线检索**。

- 离线索引：
    - **处理相似问：** 用户录入的 1 条标准问及其 200 条相似问，每一条都通过 BERT 转化为一个向量。
    - **存储：** * **MySQL：** 存储 ID 与文本内容的映射。**Milvus：** 存储 `(ID, Vector)`。
- 在线索引：
    1. **向量化：** 用户输入新问题 Q，调用 BERT 服务得到向量 V_q。
    2. **向量检索 (Milvus)：** 使用 `Search` 接口，在向量数据库中进行检索。