# LangSmith Prompt 工程化平台 - 视频讲稿

时长：14-15分钟  
目标受众：AI 应用开发者、提示词工程师、技术团队

## 📋 内容结构与时间分配

| 部分 | 标题 | 时长 | 核心内容 |
| --- | --- | --- | --- |
| 1 | 开场与痛点 | 1.5分钟 | 提示词开发的4大痛点 |
| 2 | **金字塔设计方法论** | 2.5分钟 | **业务需求→功能要求→评估方法** |
| 3 | 工程化平台理念 | 1.5分钟 | 三大理念、功能分类 |
| 4 | 核心功能演示 | 6.5-7分钟 | Prompt管理、Dataset捕获、评估系统 |
| 5 | 价值总结 | 1.5分钟 | 效率对比、核心优势 |
| 6 | 总结与展望 | 1分钟 | 要点回顾、开始使用 |

**核心亮点**： - 💡 **方法论先行**：先讲如何设计好的提示词（金字塔思维） - 🔧 **工具支撑**：再讲如何工程化管理提示词（自动化平台） - 📊 **价值明确**：最后用数据说话（10倍效率提升）

## 第一部分：开场与痛点（1.5分钟）

### 开场白

大家好，今天我要分享一个企业级提示词工程化的完整解决方案。在正式介绍之前，我想先问大家几个问题：

**你是否遇到过这些问题？**

1. **提示词管理混乱**
   * 提示词散落在代码各处，难以维护
   * 想优化提示词，但不知道从哪里开始
   * 团队成员各自维护不同版本，无法统一
2. **测试成本高昂**
   * 每次修改提示词都要手动构建测试用例
   * 想在 LangSmith Playground 中测试，却要不停地复制粘贴参数
   * 测试数据和真实场景脱节，测试结果不可靠
3. **团队协作困难**
   * A 优化了提示词，B 还在用旧版本
   * 不知道当前版本的质量如何
   * 回滚版本很麻烦，没有版本管理
4. **质量无法量化**
   * 不知道提示词改进后效果如何
   * 缺乏标准化的评估流程
   * 上线后才发现问题

### 我们的思考

这些问题的本质是：**提示词开发缺乏工程化的标准流程**。

现在，设计好提示词上线后，实际业务流程会自动采集和保存所有测试案例，回到平台即可看到详细的测试结果和对比分析，无需人工整理。通过可视化界面，你可以像调试代码一样实时修改提示词、调整参数，所有环节都全自动化、一目了然，极为高效便捷。除此之外，平台还支持一键 evaluate，可以自动评估你的提示词是否符合预期要求，在每一次迭代时也会自动检测是否影响了之前的标准，从而让优化和回归测试变得更简单可靠。

## 第二部分：提示词设计的金字塔思维（2.5分钟）

在介绍工具之前，我想先分享一个核心方法论：**如何设计一个好的提示词**。

这是整个工程化平台的基础。如果提示词本身设计得不好，再强大的工具也无法帮你。

### 金字塔思维：自上而下的设计方法

我们采用金字塔思维，从业务需求出发，层层拆解：

┌─────────────────┐  
 │ 第一层：业务需求 │  
 │ What & Why │  
 └────────┬────────┘  
 │  
 ┌────────▼────────┐  
 │ 第二层：功能要求 │  
 │ How │  
 └────────┬────────┘  
 │  
 ┌──────────────┼──────────────┐  
 ▼ ▼ ▼  
 ┌─────────┐ ┌─────────┐ ┌─────────┐  
 │ 输入要求 │ │ 输出要求 │ │ 约束条件 │  
 └─────────┘ └─────────┘ └─────────┘  
 │ │ │  
 └──────────────┼──────────────┘  
 │  
 ┌────────▼────────┐  
 │ 第三层：评估方法 │  
 │ Measure │  
 └─────────────────┘

让我用一个实际案例来说明。

### 案例：设计”行业报告生成”提示词

#### **第一层：业务需求分析**

**核心问题**： - **What**：我们要做什么？→ 生成专业的行业分析报告 - **Why**：为什么需要？→ 帮助业务团队快速了解行业动态 - **Who**：给谁用？→ 企业决策者、分析师 - **When**：什么场景？→ 制定战略、投资决策、市场研究

**明确业务目标**：

用户输入："生成 2024 年人工智能行业报告"  
期望输出：一份结构化、数据支撑、专业严谨的分析报告  
核心价值：节省 80% 的人工撰写时间，同时保证质量

**关键洞察**： - 不是写一篇泛泛的文章 - 而是生成一份**可以直接用于决策**的专业报告 - 必须有数据、有洞察、有结论

#### **第二层：功能要求拆解**

有了业务需求，我们开始拆解具体要求。分三个维度：

##### **2.1 输入要求（Input）**

需要哪些信息？  
 - 主题（topic）：必需，如"人工智能"  
 - 年份范围（year\_range）：必需，如"2023-2024"  
 - 报告风格（style）：可选，formal/casual  
 - 分析深度（depth）：可选，简要/详细/深入  
 - 关注领域（focus\_areas）：可选，技术/市场/政策  
 - 参考资料（search\_results）：必需，外部搜索的数据  
  
输入格式要求：  
 - 结构化参数（字典格式）  
 - 必填字段验证  
 - 默认值设置

##### **2.2 输出要求（Output）**

报告必须包含：  
 ✓ 标题：明确主题和时间范围  
 ✓ 执行摘要：3-5 句核心结论  
 ✓ 行业概况：当前状态和规模  
 ✓ 关键趋势：3-5 个主要发展方向  
 ✓ 数据洞察：关键数据点和解读  
 ✓ 挑战与机遇：问题和机会  
 ✓ 未来展望：1-2 年预测  
 ✓ 参考来源：数据出处  
  
输出格式要求：  
 - Markdown 结构化格式  
 - 标题层级清晰（H1/H2/H3）  
 - 数据用表格或列表呈现  
 - 字数控制：2000-3000 字

##### **2.3 约束条件（Constraints）**

质量约束：  
 ✓ 专业性：使用行业术语，避免口语化  
 ✓ 客观性：基于数据，避免主观臆断  
 ✓ 时效性：引用最新数据和趋势  
 ✓ 逻辑性：结构完整，论证充分  
  
技术约束：  
 ✓ 响应时间：< 30 秒  
 ✓ Token 限制：< 4000 tokens  
 ✓ 稳定性：成功率 > 95%  
 ✓ 成本：每次调用 < 0.1 元

#### **第三层：评估方法设计**

有了明确的要求，我们需要设计**可量化的评估方法**。

##### **3.1 结构完整性评估**（规则评估器）

def structure\_evaluator(output):  
 """检查报告结构是否完整"""  
 required\_sections = [  
 "## 执行摘要",  
 "## 行业概况",   
 "## 关键趋势",  
 "## 数据洞察",  
 "## 挑战与机遇",  
 "## 未来展望"  
 ]  
   
 score = 0  
 for section in required\_sections:  
 if section in output:  
 score += 1  
   
 return score / len(required\_sections) # 0-1 分数

**评估标准**： - ✅ 通过：所有必需章节都存在（100%） - ⚠️ 警告：缺少 1-2 个章节（70-90%） - ❌ 失败：缺少 3 个以上章节（<70%）

##### **3.2 内容完整性评估**（规则评估器）

def content\_completeness\_evaluator(output):  
 """检查内容是否充实"""  
 checks = {  
 "字数达标": len(output) >= 2000,  
 "包含数据": any(char.isdigit() for char in output),  
 "有表格或列表": ("- " in output or "|" in output),  
 "有参考来源": "参考" in output or "来源" in output,  
 "段落分明": output.count("\n\n") >= 5  
 }  
   
 return sum(checks.values()) / len(checks)

**评估标准**： - 字数、数据、格式、来源、结构，5 个维度 - 每个维度 20% 权重 - 综合分数 > 80% 为合格

##### **3.3 内容相关性评估**（LLM 评估器）

@run\_evaluator  
def relevance\_evaluator(run, example):  
 """使用 LLM 评估内容是否切题"""  
 output = run.outputs.get("report", "")  
 topic = example.inputs.get("topic", "")  
   
 # 构造评估提示词  
 eval\_prompt = f"""  
 评估以下报告是否紧扣主题"{topic}"：  
   
 {output}  
   
 评分标准：  
 1分：完全偏题  
 2分：部分相关  
 3分：基本切题  
 4分：紧密相关  
 5分：完美契合  
   
 只返回分数（1-5）和简短理由。  
 """  
   
 result = llm.invoke(eval\_prompt)  
 score = extract\_score(result) / 5.0 # 转换为 0-1  
   
 return EvaluationResult(  
 key="relevance",  
 score=score,  
 comment=result  
 )

**评估标准**： - 使用 LLM 作为评审员 - 5 分制，转换为 0-1 分数 - 自动生成评审意见

##### **3.4 参数使用准确性**（规则评估器）

def parameter\_usage\_evaluator(run, example):  
 """检查输入参数是否被正确使用"""  
 output = run.outputs.get("report", "")  
 inputs = example.inputs  
   
 checks = {  
 "使用了主题": inputs["topic"] in output,  
 "使用了年份": inputs.get("year\_range", "") in output,  
 "风格匹配": check\_style\_match(output, inputs.get("style")),  
 "引用了搜索结果": check\_reference\_usage(output, inputs.get("search\_results"))  
 }  
   
 return sum(checks.values()) / len(checks)

#### **第四步：配置化管理**

将设计的要求和评估方法写入配置：

# prompts/prompts\_config.yaml  
prompts:  
 report\_generator:  
 file: report\_generator.yaml  
 hub\_name: report\_generator  
 test\_dataset: report\_generator  
   
 # 关联的评估器  
 evaluators:  
 - "structure\_evaluator" # 结构完整性  
 - "content\_completeness\_evaluator" # 内容完整性  
 - "relevance\_evaluator" # 内容相关性（LLM）  
 - "parameter\_usage\_evaluator" # 参数使用  
   
 # 评估器权重（按重要性分配）  
 evaluator\_weights:  
 structure\_evaluator: 0.2 # 基础结构  
 content\_completeness\_evaluator: 0.2 # 内容充实  
 relevance\_evaluator: 0.4 # 相关性最重要  
 parameter\_usage\_evaluator: 0.2 # 参数使用  
   
 # 质量门禁（推送前必须达标）  
 min\_quality\_score: 0.85 # 85% 以上才能推送

### 金字塔思维的核心价值

通过这个方法论，我们实现了：

**1. 从业务到技术的完整闭环**

业务需求 → 功能要求 → 评估方法 → 持续优化  
 ↑ ↓  
 └────────────── 反馈改进 ──────────────┘

**2. 可量化的质量标准** - 不再是”感觉好像不错” - 而是”综合得分 92.5%，超过阈值 85%”

**3. 团队协作的统一语言** - 产品经理：定义业务需求 - 提示词工程师：设计功能要求和提示词 - 测试工程师：实现评估器 - 所有人：查看同一份质量报告

**4. 持续优化的数据支撑**

版本 v1.0：综合得分 78%（不合格）  
 → 优化结构和逻辑  
   
版本 v1.1：综合得分 87%（合格）  
 → 推送到 Hub  
   
版本 v1.2：综合得分 93%（优秀）  
 → 继续优化细节

### 这套方法论如何融入工程化平台？

我们的平台完美支持这个金字塔思维：

1. **YAML 文件** → 承载提示词设计（第二层：功能要求）
2. **评估器注册表** → 实现评估方法（第三层：评估方法）
3. **配置文件** → 关联评估器和权重（第四步：配置化）
4. **自动评估** → 推送前质量检查（质量门禁）
5. **Dataset 捕获** → 提供真实测试数据（持续优化）

接下来，让我展示这个平台如何支撑整个方法论。

## 第三部分：工程化平台的核心理念（1.5分钟）

### 平台设计理念

我们的平台核心理念可以总结为三句话：

1. **“远程 Hub 是唯一真相源”**
   * 团队成员启动程序时，自动拉取最新版本
   * 避免版本冲突，确保团队使用统一版本
2. **“运行即捕获”**
   * 每次运行自动保存真实参数到 Dataset
   * 不需要手动构建测试用例
   * 测试数据来自真实场景，更有价值
3. **“质量门禁”**
   * 推送前自动测试，质量不达标会警告
   * 确保每次推送都经过验证

### 系统架构

让我简单展示一下整体架构：

┌─────────────────────────────────────────────────────────┐  
│ 用户层 │  
│ main.py │ 提示词 YAML │ LangSmith Web UI │  
└─────────────────────────┬───────────────────────────────┘  
 │  
┌─────────────────────────┴───────────────────────────────┐  
│ 核心模块层 │  
├──────────────┬─────────────────┬────────────────────────┤  
│ Prompt管理 │ Dataset捕获 │ 评估系统 │  
│ - 自动拉取 │ - 装饰器捕获 │ - 自动评估 │  
│ - 智能推送 │ - 自动同步 │ - 质量打分 │  
│ - 版本管理 │ - 真实参数 │ - 对比报告 │  
└──────────────┴─────────────────┴────────────────────────┘  
 │  
┌─────────────────────────┴───────────────────────────────┐  
│ 基础服务层 │  
│ LangSmith API │ Azure OpenAI │ LangGraph │  
└─────────────────────────────────────────────────────────┘

### 功能分类说明

在介绍具体功能之前，我先说明一下：

**🔵 LangSmith 原生功能**（我们直接使用）： - Tracing 追踪 - Hub 提示词存储 - Dataset 数据集管理 - Playground 在线测试 - Evaluation 评估框架

**🟢 我们的自研功能**（工程化封装）： - 自动拉取最新版本 - 自动捕获测试参数 - 智能推送（带质量检测） - 评估器注册表和配置化 - 完整的 SOP 工作流

简单来说，**LangSmith 提供了底层能力，我们在上面构建了工程化流程**。

## 第四部分：核心功能演示（6.5-7分钟）

现在让我分模块介绍核心功能，展示平台如何支撑金字塔方法论。

### 模块一：Prompt 管理系统（1.5分钟）

#### 1.1 标准化的 YAML 格式 🟢（自研）

首先是提示词的标准化管理。我们使用 YAML 格式：

# prompts/report\_generator.yaml  
version: v1.2.0  
description: "生成行业分析报告"  
  
messages:  
 - role: system  
 content: |  
 你是专业的行业分析师。  
 根据提供的信息生成结构化报告。  
   
 - role: user  
 content: |  
 主题: {topic}  
 年份: {year\_range}  
 风格: {style}  
   
 请生成报告。  
  
input\_variables:  
 - topic  
 - year\_range  
 - style

**为什么用 YAML？** - 结构清晰，易于维护 - 支持版本管理 - 可以直接推送到 LangSmith Hub 🔵

#### 1.2 自动拉取机制 🟢（自研核心功能）

这是我们的核心创新之一。看代码：

from prompts.prompt\_manager import PromptManager  
  
# 启动时自动拉取最新版本（默认开启）  
manager = PromptManager(auto\_pull=True)  
  
# 获取提示词时，自动从 Hub 拉取最新版  
prompt = manager.get('report\_generator')

**工作流程**： 1. 检查 LangSmith Hub 🔵 的最新版本 2. 对比本地版本号 3. 如果有新版本，自动下载并更新本地 YAML 文件 4. 加载并返回最新配置

**价值**： - ✅ 团队成员无需手动同步 - ✅ 启动即获取最新版本 - ✅ 避免版本冲突

#### 1.3 智能推送（带质量检测）🟢（自研核心功能）

当你优化了提示词，想要推送到 Hub：

# 推送到 Hub，带自动测试  
manager.push(  
 'report\_generator',  
 with\_test=True, # 推送前自动测试  
 create\_backup=True, # 创建版本备份  
 commit\_message='优化报告结构'  
)

**自动执行 4 步流程**：

1. **验证格式** 🟢
   * 检查 YAML 语法
   * 验证必需字段
2. **自动测试** 🟢（使用 LangSmith Evaluation 🔵）
   * 从 Dataset 🔵 加载测试用例
   * 运行评估器
   * 计算质量分数
   * 如果分数低于阈值，发出警告
3. **推送到 Hub** 🔵
   * 上传到 LangSmith Hub
4. **创建备份** 🟢
   * 保存历史版本快照
   * 支持随时回滚

**价值**： - ✅ 质量门禁，确保推送的提示词经过验证 - ✅ 自动化流程，减少人工操作 - ✅ 版本可追溯，可以随时回滚

### 模块二：Dataset 自动捕获系统（2分钟）

这是另一个核心创新。**这直接对应金字塔方法论的第三层：提供真实测试数据**。

#### 2.1 问题场景

传统方式：

1. 手动构建测试用例（耗时）  
2. 复制粘贴参数到 Playground（繁琐）  
3. 测试数据和真实场景脱节（不可靠）

我们的方案：**运行即捕获** 🟢

#### 2.2 装饰器自动捕获 🟢（自研核心功能）

在代码中使用装饰器：

from tools.capture import capture\_dataset  
from langsmith import traceable  
  
@traceable # LangSmith 追踪 🔵  
@capture\_dataset(  
 prompt\_name="report\_generator",  
 dataset\_name="report\_generator",  
 auto\_sync=True # 自动同步到 LangSmith 🟢  
)  
def generate\_report\_node(self, state):  
 # 准备参数  
 inputs = {  
 "topic": state.get("topic"),  
 "year\_range": state.get("year\_range"),  
 "style": state.get("style"),  
 "search\_results": state.get("search\_results")  
 }  
   
 # 标记要捕获的参数  
 capture\_inputs(inputs) # 🟢 自研函数  
   
 # 调用 LLM  
 result = chain.invoke(inputs)  
 return result

**自动执行**： 1. 捕获函数调用的原始参数字典 2. 保存到本地缓存 .dataset\_cache/ 3. 自动推送到 LangSmith Dataset 🔵 4. 关联 run\_id 和 metadata

#### 2.3 捕获的数据格式

{  
 "prompt\_name": "report\_generator",  
 "timestamp": "2024-10-27T14:30:52",  
 "inputs": {  
 "topic": "人工智能",  
 "year\_range": "2023-2024",  
 "style": "formal",  
 "search\_results": "根据最新数据..."  
 },  
 "metadata": {  
 "user\_query": "生成AI行业报告",  
 "prompt\_version": "v1.2"  
 }  
}

**关键点**：捕获的是**原始参数字典**，不是格式化后的文本。

#### 2.4 在 Playground 中的强大用法 🔵+🟢

**完整工作流**：

第1步：运行程序（自动捕获）🟢  
 → python main.py --query "生成AI行业报告"  
 → 参数自动保存到 Dataset  
  
第2步：在 LangSmith Playground 测试 🔵  
 → 打开 Playground  
 → 选择 "report\_generator" Dataset  
 → 看到所有自动捕获的真实测试用例  
  
第3步：切换版本对比 🔵  
 → 下拉选择 v1.0、v1.1、v1.2  
 → inputs 参数自动保持不变  
 → 实时查看不同版本的输出差异  
  
第4步：选择最优版本推送 🟢  
 → 在 Playground 确定最优版本  
 → 本地更新 YAML  
 → 推送到 Hub

**价值**： - ✅ 测试数据来自真实运行，更可靠 - ✅ 无需手动复制粘贴参数 - ✅ 版本对比一目了然 - ✅ 节省 95% 的测试准备时间

### 模块三：评估系统（1.5分钟）

**这完美实现了金字塔方法论的第三层：评估方法可配置化**。

#### 3.1 评估器注册表 🟢（自研）

我们构建了一个灵活的评估器系统：

# evaluation/evaluators/\_\_init\_\_.py  
EVALUATOR\_REGISTRY = {  
 # 通用评估器  
 "structure\_evaluator": ReportEvaluators.structure\_evaluator,  
 "content\_completeness": ReportEvaluators.content\_completeness,  
   
 # 参数提取专用  
 "parameter\_extraction": ParameterEvaluators.parameter\_extraction,  
 "field\_type\_checker": ParameterEvaluators.field\_type\_checker,  
}

#### 3.2 配置化评估 🟢（自研）

在配置文件中指定评估器：

# prompts/prompts\_config.yaml  
prompts:  
 report\_generator:  
 file: report\_generator.yaml  
 hub\_name: report\_generator  
 test\_dataset: report\_generator  
   
 # 指定评估器  
 evaluators:  
 - "structure\_evaluator"  
 - "content\_completeness"  
 - "relevance\_evaluator"  
   
 # 评估器权重  
 evaluator\_weights:  
 structure\_evaluator: 0.2  
 content\_completeness: 0.3  
 relevance\_evaluator: 0.5  
   
 # 质量阈值  
 min\_quality\_score: 0.85

#### 3.3 运行评估 🟢+🔵

from prompts.prompt\_manager import PromptManager  
  
manager = PromptManager()  
  
# 评估提示词质量（使用 LangSmith Evaluation API 🔵）  
result = manager.evaluate\_prompt('report\_generator')  
  
print(f"质量评分: {result['quality\_score']:.2%}")  
print(f"测试用例数: {result['total']}")

**评估报告示例**：

============================================================  
提示词质量评估报告  
============================================================  
提示词: report\_generator  
测试用例数: 15  
  
评估结果:  
------------------------------------------------------------  
 structure\_valid : 95.0% (权重: 0.2)  
 content\_completeness : 88.0% (权重: 0.3)  
 relevance\_score : 92.0% (权重: 0.5)  
------------------------------------------------------------  
 综合质量分数: 90.9% ✅  
  
状态: 通过（阈值: 85%）

**价值**： - ✅ 质量可量化 - ✅ 自动化评估 - ✅ 灵活配置评估器

### 模块四：完整工作流（1.5-2分钟）

现在让我演示一个完整的工作流，展示**金字塔方法论如何在实际中运作**：

#### 场景：开发者 A 优化提示词

# 步骤1：修改本地 YAML 文件  
vim prompts/report\_generator.yaml  
# （优化提示词内容）  
  
# 步骤2：本地测试  
python main.py --query "生成2024年AI行业报告"  
# 🟢 自动捕获测试参数到 Dataset  
# 🔵 LangSmith 追踪整个流程  
  
# 步骤3：查看捕获的数据  
python tools/capture.py --list  
  
# 步骤4：在 Playground 验证 🔵  
# （打开 LangSmith Playground）  
# （选择 Dataset，对比版本）  
  
# 步骤5：推送到 Hub  
python -c "  
from prompts.prompt\_manager import PromptManager  
manager = PromptManager()  
manager.push('report\_generator', with\_test=True)  
"  
# 🟢 自动测试 → 质量检查 → 推送到 Hub 🔵

#### 场景：开发者 B 自动同步

# 开发者 B 第二天来上班  
python main.py --query "生成报告"  
  
# 自动发生：  
# 🟢 检查 Hub 最新版本  
# 🟢 发现有新版本，自动下载  
# 🟢 更新本地 YAML 文件  
# ✅ 使用最新版本生成报告

**协作流程**： - A 推送 → Hub 更新 → B 自动拉取 - 全程自动化，无需手动通知 - 团队始终使用统一版本

## 第五部分：价值总结（1.5分钟）

### 效率提升对比

让我用数据说话：

| 传统方式 | 工程化方式 | 节省时间 |
| --- | --- | --- |
| 手动构建测试用例 | 运行即捕获 | **90%** |
| 复制粘贴参数 | 自动推送到 Dataset | **95%** |
| 手动通知团队更新 | 自动拉取最新版本 | **100%** |
| 手动记录版本 | 自动备份 | **100%** |
| 手动测试评估 | 推送时自动测试 | **85%** |

**总体提升：2小时 → 10分钟** 🚀

### 核心优势

**1. 自动化** - 自动拉取、自动捕获、自动同步、自动评估 - 减少 90% 的手动操作

**2. 标准化** - 统一的 YAML 格式 - 标准化的工作流程 - 可复制的最佳实践

**3. 工程化** - 版本管理 - 质量门禁 - 团队协作

**4. 可视化** - 清晰的质量分数 - 版本对比报告 - 追踪完整流程

### 适用场景

这套方案特别适合：

✅ **企业级 AI 应用开发** - 需要团队协作 - 提示词数量多 - 质量要求高

✅ **提示词工程师** - 需要频繁优化提示词 - 需要对比测试效果 - 需要量化质量提升

✅ **AI 应用产品化** - 需要持续优化 - 需要版本管理 - 需要质量保障

## 第六部分：总结与展望（1分钟）

### 核心要点回顾

让我总结今天分享的核心要点：

**1. 金字塔设计方法论** - 业务需求 → 功能要求 → 评估方法 - 从 What & Why 到 How 再到 Measure - 可量化、可配置、可迭代

**2. 三大工程化理念** - 远程 Hub 是唯一真相源（自动拉取） - 运行即捕获（自动测试数据） - 质量门禁（推送前验证）

**3. 两大创新功能** - 🟢 自动拉取+智能推送（Prompt 管理） - 🟢 装饰器捕获+自动同步（Dataset 管理）

**4. 一套完整 SOP** - 设计 → 开发 → 测试 → 评估 → 优化 → 部署 - 全流程自动化

### 技术栈说明

**基础能力（LangSmith 🔵）**： - Tracing、Hub、Dataset、Playground、Evaluation

**工程化封装（我们 🟢）**： - PromptManager、DatasetCapture、EvaluationRunner - 自动拉取、自动捕获、智能推送 - 配置化管理、SOP 流程

### 开始使用

# 克隆项目  
git clone <your-repo>  
cd Langsmith-prompt-pipeline  
  
# 安装依赖  
pip install -r requirements.txt  
  
# 配置环境变量  
cp .env.example .env  
# 编辑 .env 填入 API Keys  
  
# 运行第一个示例  
python main.py --query "生成AI行业报告"  
  
# 查看 LangSmith 控制台  
# https://smith.langchain.com/

### 结尾

提示词工程化不仅仅是技术问题，更是效率和质量的问题。

希望这套方案能帮助你的团队： - ✅ 提升 10 倍开发效率 - ✅ 保障提示词质量 - ✅ 实现团队无缝协作

如果你对这个项目感兴趣，欢迎： - ⭐ Star 项目 - 🔗 查看完整文档 - 💬 提交 Issue 和 PR

感谢大家的观看！

## 附录：演示脚本建议

### 建议的视频结构

**开场（30秒）** - 展示项目 Logo/架构图 - 抛出痛点问题

**第一部分（2分钟）** - PPT 展示痛点场景 - 动画演示传统方式的繁琐流程

**第二部分（2分钟）** - 展示架构图 - 标注 LangSmith 🔵 和自研 🟢 功能

**第三部分（7-8分钟）** - **屏幕录制**：实际操作演示 - 展示 YAML 文件 - 运行程序（显示自动捕获） - 打开 LangSmith Playground - 演示版本切换 - 运行推送命令 - 展示评估报告

**第四部分（1.5分钟）** - PPT 展示对比表格 - 动画展示效率提升

**结尾（1分钟）** - 展示项目链接 - Call to Action

### 关键可视化建议

1. **流程图动画**
   * 传统方式 vs 工程化方式
   * 突出自动化环节
2. **屏幕录制**
   * 分屏展示：左边代码，右边 LangSmith UI
   * 加速播放重复操作
   * 关键步骤加标注
3. **对比表格**
   * 使用进度条动画
   * 突出节省的时间
4. **架构图**
   * 用颜色区分 LangSmith 🔵 和自研 🟢
   * 箭头动画展示数据流

### 演讲技巧

1. **语速控制**
   * 关键概念放慢
   * 演示部分可以加速
2. **互动提问**
   * 开场抛问题
   * 引导观众思考
3. **重复强调**
   * 核心理念重复 2-3 次
   * 关键数字突出显示
4. **故事化**
   * 用”开发者 A/B”的故事串联
   * 增加代入感