人工智能基础期末大作业技术报告

刘英哲[1] **李靖昕**[2] **张霖泽**[3]

1 信息科学技术学院，应用物理学，2300012753
2 信科信息技术学院，计算机，2400013201
3 信息科学技术学院，计算机科学，2300013724

Abstract

在当代注意力紧张的数字内容生态中，品牌内容营销面临大锅饭个性化需求与传统精益求精的内容生产模式之间的矛盾，这导致了策略脱节、创意枯竭、生产效率低下等问题。为应对此挑战，本文实现了一个品牌内容营销策划系统。该系统基于 Multi-Agent 协作与工具调用，旨在自动化内容营销的全流程，让机器完成可计算的、基本的的部分，从而提升人类高效生产高质量内容营销的能力。其核心由三个功能解耦的 Module 构成: 1）市场洞察与策略定位 Agent，负责整合多源数据，执行深度竞品分析与受众画像构建，并研究基本的行业趋势；2）内容创意孵化与多维形式策划 Agnet，在既定战略框架下，进行规模化的基础创意生成、内容主题规划与跨平台分发策略设计；3）视觉传达设计与发布执行细化 Agent，负责将抽象的品牌“人设”转译为具体的视觉识别系统（Social VI），设计相应的主视觉草图，并生成包含具体发布策略与 KPI 的可执行方案。本系统通过整合 LLM 与外部 API，实现了从初始信息输入到生成完整、数据驱动且高度定制化的营销策划包的端到端自动化。实验结果表明，该系统能显著缩短策划周期，并有效提升内容生产的效率。

1. 需求分析

本系统针对的是当前数字营销生态下品牌方面临的深刻、复杂挑战。品牌的营销需求不仅是功能列表的罗列，更是对行业趋势、用户倾向的系统性回应。

目前的产品数字营销面对的问题有:

- **信息过载与注意力稀缺**。用户每天被社交媒体、新闻、广告等海量内容淹没，其注意力已成为最宝贵的稀缺资源。品牌的内容如果不能在最初的几秒内抓住用户眼球、精准触及其需求或兴趣点，就会被瞬间滑过。这要求内容不仅要制作精良，更要具备极高的相关性和价值密度。
- **竞争白热化与策略同质化**。几乎所有行业的市场都已是红海，竞争者众多。更严峻的是，随着营销知识的普及，许多品牌采用了相似的策略和打法，导致营销手段趋同。若无基于深度市场洞察的独特品牌定位和差异化叙事，品牌极易陷入价格战和流量战的同质化泥潭，难以沉淀用户忠诚度，无法构建真正的品牌护城河。
- **生活节奏与内容产出节奏的失衡**。用户的现代生活节奏快、时间被切割得非常零碎。内容消费发生在通勤、排队、睡前等碎片化时间。这种消费习惯极度偏爱“短、平、快”的快餐式内容。为了迎合这种消费习惯并持续“喂养”平台算法，品牌面临着巨大的内容生产压力。这种压力不仅消耗大量资源，更容易导致团队创意枯竭、动作变形，为了数量牺牲质量，最终导致产出大量低质量内容。

为此，本系统考虑如何在保持价值对齐[1]的前提下，

[1]每一张图、每一段文案，都能够向上追溯到最初的商业目标和品牌价值体系，保证策略与执行的协调统一，维护与核心用户群体的想关心

利用生成式 AI，将过去需要数小时甚至数日的创意构思、文案撰写、视觉原型设计等环节，压缩至分钟级别，以此提升了单个内容元件的生产速度，方便后续人工更高效率的分析处理。即，本系统希望从“内容创作者”到“内容资产管理者”的范式转移，使质量与效率不再是不可兼得的对立面。品牌可以利用该系统，在不显著增加人力与预算的前提下，维持在多个平台的高频次、高质量曝光，将团队从重复性的生产劳动中解放出来，专注入更高层次的策略与创意把控。

2. 技术选型

经过对提供的多种框架进行初步了解和尝试后，我们选择了 camel-ai 作为我们的实现框架。而 camel-ai 中也提供了多种互动模式：单智能体互动 ChatAgent，两个智能体角色扮演 RolePlaying，多智能体工作团队模式 WorkForce 等。

2.1. Agent1

完整代理的第一部分 agent1 的主要任务是通过命令行接受并拓展用户提供的材料。

基于大模型生成的 agent1 任务，发现需要将主任务拆分为多个子任务，交给多个智能体完成。这个过程中由于大模型可能存在幻觉、遗漏要点等问题，需要人类对 agent1 得到生成结果进行审核后才能进入后续的生成。初步讨论得到了三种方案：

1. 使用多个 ChatAgent 串联，每个 ChatAgent 接受初始输入以及之前 agent 的工作成果来生成输出；
2. 类似方案 1，但使用 RolePlaying 代替 ChatAgent；
3. 采用 WorkForce，将每一个子任务分配给一个智能体，多个智能体共同工作以生成结果

在初步尝试后，发现：
由于提供的 api 对应的模型”deepseek-ai/DeepSeek-R1” 不支持异步推理，无法使用 WorkForce；
方案 1,2 之间效果类似。其中方案 2 由于每个子任务都有扮演用户的智能体，产出结果大部分情况下都优于方案 1，但偶尔也会出现凭空捏造需求的情况。且耗时显著长于方案 1。

因此最终选择使用方案 1 完成 agent1 的实现

2.2. Agent2

Agent 2 的主要任务是将 Agent 1 给出的较为宏观的品牌策略转化为可执行化的发布策略，包括内容创意、平台适配等。作为链接“战略”和“执行”两部分的桥梁，本部分关键并非在于单一任务的深度，而在于对前一模块的输出信息整合、关联、发散的综合处理，并将结果稳定地投喂给下一模块的能力。

基于以上分析，并考虑到 deepseek-R1 模型对 WorkForce 的不适配性，最终得到以下两种主要架构：

1. **单一巨型智能体模式**：构建一个单一、巨大的 Agent，通过前一模型输出转换而来包含所有子任务指令的 SuperPrompt 一次性生成结果。
2. **流水串联智能体模式**：将 Agent 2 的总任务拆解为一系列逻辑上前后关联的子任务（创意风暴 -> 系列规划 -> 平台适配 -> 深度策划 -> 测试文案 -> 日程编排），每个子任务由一个独立的 ChatAgent 负责。每个 Agent 的输出作为下一个 Agent 的核心输入，并增加相关的 Agent 的输出和前一模块输入，以形成一条有效且明晰的“内容生产流水线”。

在初步尝试实现的测试过程中，我们发现：

- **单一巨型智能体模式的局限性**：当 Prompt 过于冗长或者结构复杂时，单一模型常出现指令遗忘、重点漂移、输出格式不合要求、重要内容信息丢失等现象。如某次模型出色地完成了创意内容生成（这一部分任务靠前）却忽略了后续发布日程（这一部分任务靠后）的格式要求。此外，这种“一锤子买卖”的模式导致整个流程缺乏可控性，一旦某个环节输出不理想，常常需要重新运行整个庞大的任务，而输出的不

确定性导致若再次不理想，就要不断重复这一过程。此外，之前生成的中意的创意，之后可能又消失掉了；而之后生成的中意的策划，却和之前中意的创意无关，导致“见到了最好的点子却不能适配最好的策划”，即便最终生成了一个可用的整体架构，却心有不甘患得患失，整体效果不佳的同时还要反复运行庞大的任务，tokens 和时间成本都很高。

- **流水串联智能体模式的优势**：在方案 1 受挫后我查阅资料并转向了方案 2——将宏大的模块化任务按内容和逻辑分拆为一系列前后相连、目标明确的子任务并通过智能体串联来逐一攻破。实验证明这种架构确实优于单一智能体，主要表现在以下几个方面：
  - **专注性与质量提升**：相较单一智能体，串联智能体中每个 ChatAgent 只需专注于分配给其的具体任务，例如负责 BrainStorm 的 Agent 可以专注于发散性思考，而无需考虑其他任务；与此同时负责 PublishCalendar 的 Agent 则可以专注于按照所需要求格式化输出。这种“每个部分只需要做好自己分内的事”的专注性任务分配显著提搞了每一环节结果产出的质量和稳定性。
  - **可控性与可调试性**：相较单一智能体的不稳定性问题，多智能体串联使我们能检验每步智能体的输入和产出。一旦不符预期，我们可以直接并仅调整该特定 Agent 而无需更改整体框架。这种形式极大地便利了程序的调试，提高了开发和优化效率。
  - **逻辑清晰性与思维流程化**：相较单一智能体，多智能体串联模拟了人类在内容策划过程中的思维流程：先是生发出创意点，再将这些点子按相关性归类，然后分别按照内容调整、平台适配、文案撰写的方式匹配形式，最后编排到日程表中。这种清晰的逻辑结构使得代码流程化和模块化，更易于理解和维护。

因此，我们最终确定**采用方案 2**，即流水串联智能体模式来构建 Agent 2 以充分发挥其作为 Agent 1 和 Agent 3 的桥梁作用。

另外，在 Agent 2 的流程中我统一使用了 deepseek-R1，原因主要在于以下几个方面：

1. deepseek-R1 在中文语境下的理解和生成能力表现出色，能准确把握中文品牌资料的细微差别并产出符合国内公司背景的创意；

2. deepseek-R1 上下文窗口足够支撑每个子任务的 Prompt 长度；

3. deepseek-R1 在响应速度、tokens 成本和运行时间上达到良好平衡。

2.3. Agent3

Agent3 负责基于前序 Agent 输出的策略与创意，构建社交媒体视觉识别系统，完成多平台视觉模板设计、AI 辅助视觉内容生成，同时细化发布策略、优化文案与视觉匹配度，并制定效果追踪指标体系。此模块需融合品牌调性、内容特性与受众视觉偏好，同时兼顾发布推广策略，对多源信息进行整合、转化与细化输出。

在设计过程中进行了如下尝试：

1. **尝试 dify 框架**。由于缺少自由性，往往需要讲所有数据一次输入，中间结果的存储与最终结果的输出过于受限。

2. **串联多个 Agent 组成线性数据流动，并直接利用一个 Agent 生成多项任务的 KPI、主视觉**。此方案，导致大量的冗余信息被输入给 LLM，导致生成质量差，幻觉高且不同任务重复性强，整体生成质量低下；导致不同 LLM 请求中 KV 重复计算，资源浪费严重

3. **采取 workforce**。采用 camel 框架下的 workforce 过程，设定评判组与工作组，相对而言提升里生成质量，但是产生了新的问题：由于 SiliconFlow 的 API 不支持异步调度，必须采用个人的 API；在 reply 过程中上下文长度过长，逻辑性下降，导致最后常常出现 time out 的情形，

系统稳定性不足且总体耗时过长；workforce 输出信息用于生成主视觉，任然做不到根据 task 不同生成特化的主视觉图案的期望

4. **发挥 python 程序的灵活性，手动维护数据流，先拆分出 task 再对每个 task 分别进行主视觉生成、内容发布策略规划**。这样，不仅能灵活分配数据流向，减少不必要的计算开销，耗时更少；还能生成更高质量的主视觉图像以及更有针对性的发布策略。

基于上述尝试，最后选定了 python 代码 +camel 框架 + 外观模式（即**尝试 4**）的设计思路，将数据输入输出拆分到 5 部分：流行视觉趋势分析、视觉系统设计、多层次 KPI 构建、单任务发布策略处理、单任务主视觉生成，也把整体流程拆分成五个模块分别实现。

关于文生图 API 选型的尝试过程：

1. 尝试了 stable_diffusion3.5、hunyuanDiT、gemini 文生图、wanx2 等可选模型
2. 排除 sd3.5，由于需要本地部署；
3. 排除 hunyuan，由于 API 调用需要的信息过多
4. 排除 gemini，因为网络连接问题以及图中字符生成质量差的问题

最后，选择调用 wanx2 作为文生图的 API

3. 实现细节

3.1. 依赖管理

为了方便用户部署这个 agent，采用了本地创建 venv，并使用 requirements.txt 给出所需依赖的方式。具体见 README

3.2. 使用方法

为了尽可能提高代理的自由度，尤其是人类提供材料的多样性，代理的文件读取方式设计为人类提供一个.json 文件作为材料列表，格式如下：

```
{
  "files": [
    {
      "path": "./path/to/first/file",
      "comment": "comment of first file",
      "type": "type of first file"
    },
    ...
  ]
}
```

在此条件下人类需要提供的仅有输入文件列表的.json 文件，因此可以直接采取命令行交互：

```
(Myvenv) agentdir > python agent_general.py
    -i path/to/input.json
```

代理的三个部分 agent1,agent2,agent3 之间也采取同样的方式进行对接

3.3. Agent1

根据之前确定的大致方案，需要创建 6 个 ChatAgent. 其中前五个为实现子目标的代理，最后一个负责整合输出报告。

以子任务 4 代理为例，构建子任务 4 代理的代码：

`agent1.py line 137-152`

其中`files_info`即为人类提供的传入文件信息，`model`为预设好参数的模型

代理构建完成后即开始按序依次向 6 个 agent 发送请求。发送请求时提供原始任务以及前置任务结果：

`agent1.py line 266-287`

这一过程中无需人类互动。在这一部分完成后，会将此回答传递给一个新的 agent，此 agent 专门负责根据人类的评价修改/增添/删减报告内容。
同时，为了方便将最终的报告拆分为可以传递给 Agent2 的格式，在输出报告时我们要求其第 i 篇报告以”report_i_start” 开始，以”report_i_end” 结束。通过这个提示词，可以直接使用在回答中查找标识符的方式实现输出的拆解以及按照对接格式 (见 3.2) 重新构建

3.4. Agent2

基于“流水串联智能体”的技术选型，Agent 2 通过一个核心类 ContentCreativityAgent 实现。其通过一系列子任务和细化方法，模拟从战略到创意的完整流程，并产出投喂给下一模块的结果。

3.4.1. 数据流与初始化

Agent 2 作为链接 Agent 1 和 Agent 3 的桥梁，其输入始于对 Agent 1 产出的承接。按照 Agent 1 提供的结果，我们采用解析 json 文件读取文档作为输入，并按照类似方式产出一个输出结果文件夹，内含六个文件和便于下一 Agent 读取的 json 文档。

- **输入**：Agent 2 的输入依赖于 Agent 1 生成的 `agent1_outputs.json` 配置文件。该 JSON 文件中包含了 Agent 1 生成的文件路径及描述。
- **初始化**：在 ContentCreativityAgent 的构造函数 `__init__` 中，程序首先调用 `utils.py` 中的辅助函数 `load_file_config` 和 `load_files_from_config`读取 JSON 配置，并根据路径加载文件内容，最终将信息存入 `self.input_data` 字典中。这种设计实现了输入模块的去格式数据化，使得 Agent 2 不关心上游文件的具体数量和名称，只关心其内容，具备良好的可读性、数据性和扩展性。
- **输出**：为与 Agent 3 对接，`__init__` 方法初始化一个空列表 `self.output_files`。在后续每个文件生成后，程序都会将生成文件的基本信息追加到此列表中。整个流程结束时，调用 `_write_output_json` 方法，将该列表序列化为 `agent2_outputs.json`，完成向 Agent3 的传递。

3.4.2. 核心流水线实现

`execute_planning_flow` 方法是 Agent 2 的主控函数，定义了全部内容策划的执行顺序。本主控函数由六个核心步骤构成，每个步骤都由一个独立的 ChatAgent 实例驱动，每个实例都配备有针对其特定任务而精心设计的 Prompt，以获得最佳效果。

以流水线中的第一个任务为例，其实现细节`run_ideation_and_filtering` 如下：

```
def run_ideation_and_filtering(self) ->
    List[Dict[str, Any]]:
formatted_inputs = self.
    _format_inputs_for_prompt()
prompt = f"""
**核心任务：创意风暴与初步筛选**
（prompt较长，省略，可参见源代码）
"""
response = self._execute_prompt(prompt)
...  （较长省略，可参见源代码）
```

后续方法除提示词和输入来源外皆遵照此**准备上下文 -> 构建指令 -> 调用模型 -> 解析输出**模式

3.4.3. 动态工具调用：集成搜索能力

单纯由大模型生成的创意虽丰富以至于天马行空，却大多缺乏现实生活相关的内容，难以取得平台流量。因此，为了增强内容的实用性，在 Agent 2 的流水线中，我们还集成了一个关键的动态工具调用环节——`_real_seo_tool_with_llm` 函数。

- **实现方式**：该函数本质上是一个“Agent 中的 Agent”。它通过构建一个专门面向搜索优化任务的 Prompt，复用 `_execute_prompt` 方法来调用 LLM，为给定的内容主题生成一系列关键词。
- **集成节点**：在为旗舰内容撰写详细大纲之前，程序会先将内容的创意标题传递给 `_real_seo_tool_with_llm`。
- **价值体现**：获取到的关键词列表随后会被注入到 `develop_flagship_content` 的 Prompt 中。通过这一步，模型输出不仅更有创意，而且具备了数据驱动、易被搜索引擎发现的潜力，是连接创意策划与实际营销效果的重要一环。

该工具的实现代码如下：

```
def _real_seo_tool_with_llm(self, topic,
    num_keywords) -> List[str]:
seo_prompt = f"""是一名顶级的搜索引擎优化专
    家...(完整的提示词详见源代码)"""
```

```
response = self._execute_prompt(seo_prompt)
keywords = [... for ... if ...]
cleaned_keywords = [re.sub ... ]
return cleaned_keywords
```

通过这种方式，Agent 2 不再是一个相对封闭的系统，而是能够自调用内部工具，动态捕捉任务相关的、更专业的信息，从而较大地提升了最终输出方案的实用价值。

3.5. Agent3

3.5.1. 数据预处理

```
def contents_calendar_to_list(
    marketing_plan_text) -> dict
def extract_content_between_delimiters(
    input_str: str, N: int) -> list[str]
def process_and_save_delimited_blocks(
    txt_path: str,
    N: int,
    output_dir: str,
    output_names: list,
    output_json_name: str = 'agent1_outputs
        .json'
)
def JsonToNL(content)
def deal_data_for_agent_3(
    needed_data_for_agent_three: dict)
```

主要实现了内容日历解析、报告内容块提取与拆分、JSON 到自然语言转换等功能。它通过调用大模型 Agent 自动将营销计划文本转为结构化任务列表，并支持将长文本报告按标记拆分为多份文件，便于后续协作。

3.5.2. 纵览数据流动

每一个模块都由具体功能类和 facede 函数构成，具体数据流动如图 Figure 1.

3.5.3. 单个模块为例

以文生图 $(agent3_modules/m3_generate_image.py)$ 为例：该模块根据内容日历条目和品牌 VI 系统，自动生成主视觉图片的英文 prompt，

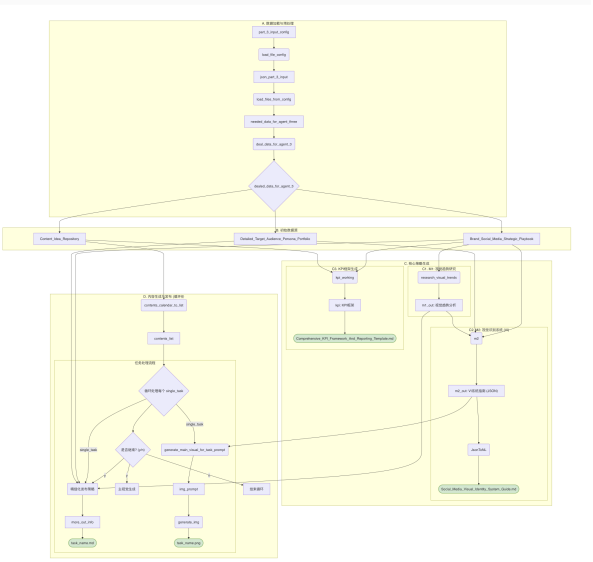

Figure 1. 粗粒度的数据流动图

并调用 API 生成图片，支持人类反馈迭代优化；而接口只是函数 *generate_img*、*generate_main_visual_for_task_prompt*。采用外观模式设计，通过顶层函数作为统一入口，封装了 *ImageGenerator* 的复杂迭代生成逻辑与底层工具函数的调用细节，向用户暴露简单接口，从而降低了 Client 复杂度。

4. 评估对比

4.1. 不同组织架构的对比

考虑抽象程度与多模态，比较 Agent1 与 Agent3：

Agent 的代码书写复用多，逻辑简明清晰；Agent3 不同模块独立，复用不足，但是数据流动更高，便于多任务并行。

Agent1 实现子任务自动串行 + 最终报告生成的全流程自动化，不同任务抽象为相同 model 不同 task，自动传递前序任务结果，直至最终报告输出；代码结构较为紧凑，通过循环和统一接口简化了大部分书写代码的冗余。

反而 Agent3 不同模块过于独立，无法用简单的方法进行串联的书写，但是数据传递相对自由，方便

在

Agent1 的文本流易于抽象和优化；Agent3 的多模态流则难以统一抽象，导致优化空间受限。

Agent1 处理的文本，所有中间结果都可以抽象成字符串活结构化文本用于复用。

而在设计 Agent3 时候需要设计文本、结构化数据、图片等多模态数据，导致优化空间受限、重复 prefill 的内容较多。难以抽象为同一中间态，每个模态输入的方式不同，难以完全自动抽象串行

Agent3 在自动化和并行化层面相对更优，但由于多模态和多模块协作，LLM 重复计算和代码冗余更为显著。这是多模态智能系统在工程实现中面临的难题，也是未来进一步提升效率和降低成本的关键挑战。

4.2. 不同 temperature 下模型生成内容比较

以 Agent 2 的输出为例，我们将 0.5-1.2 划分为低温区、中温区和高温区三个部分，并按照步长 0.1（在区间 0.7-1.0 之间细化为 0.05）对不同温度下模型生成内容进行比对分析

- **低温区** (temp = 0.5 - 0.7)

  输出特征:

  - **结构严谨**：这一温区模型严格遵守 Prompt 定义，如深度策划产出内容严格遵照三段式层间结构，产出内容标准化且逻辑清晰。
  - **内容聚焦**：主题严格源于输入，标题直接而明确。如 temp=0.5 的策划案中“程序员能量补给黄金清单”。
  - **语言务实**：输出的语言风格偏向事实陈述和科学解释，提供具体的数据支撑，强调实用价值。如“这些零食让你专注于代码”
  - **创意保守**：这一温区内模型输出生发的创意点相对保守，但实用性强。模型的输出“坚果”、“黑巧克力”、“自制能量棒”等都是常见的闲时充饥零食（我们的策划是程序员、零食相关的内容宣传）。

  结论：低温区间内产出的内容质量相对较高，内容可靠，逻辑性强。这一温度区更适合**生成高度精确性和结构性的内容**，例如策划案等稿件相关内容，或对已确定创意的执行。

- **中温区** (temp = 0.7 - 1.0)

  输出特征:

  - **temp = 0.7 - 0.8**:
    - **营销型内容开始出现**：产出的文案中出现了“终极”、“黑科技”等吸睛的词。
    - **创意性微创新**：核心仍然是能量补给，但出现了“我们的能量棒三重奏配方”,“20-20”护眼法则等微创新具体细节概念。
  - **temp = 0.8 - 0.9**:
    - **新形态概念开始出现**：模型不再局限于食物（输入内容）本身，而产出了泛化的概念，如除能量补给外还包括了“散步提神”等广相关内容。
    - **主题升维**：模型第一次将输入文档中两个较为独立的方面“身体能量”和“编程效率”连携在一起，实现了新的创意。
  - **temp = 0.9 - 1.0**:
    - **隐喻和类比开始出现**：模型开始使用修辞手法，例如将“BUG”、“底层”等程设术语融合到能量补充的主题中：“解码程序员能量崩溃的底层 BUG”等。
    - **尝试贴近目标受众**：模型试图迎合内容背景，如上例中模型试图迎合程序员的思维模式和语言“套近乎”宣传。

  小结:

  - temp = 0.7 - 0.8:

    相较低温区，此温度区间在保持要求结构的基础上，增加了内容的吸引力和趣味性，适用于需要安全“创意增强”的文档中。
  - temp = 0.8 - 0.9:

    这一温度区间开始展示模型的联想能力：模型不再是刻板地根据输入内容生成千篇

一律的策划案，而是开始尝试将输入内容联系起来，在连接点处做创新和改变。

- temp = 0.9 - 1.0: 中温区的最高温度部分在尽量不改动要求的输入框架内，实现了较为安全的内容中最有新意、相对更易使受宣传目标产生共鸣的方式产出内容。

结论:

中温区是**创意性营销内容的适温区**。其在保证内容不偏离主题的同时，随温度升高尽可能产出具有创新力和吸引性的内容，模型逐渐从“策划执行者”变为“创新设计师”

- **高温区** (temp = 1.0 - 1.2)

  输出特征:

  - **结构松散**: 随温度进一步升高，模型开始偏离 (temp=1.1) 甚至完全抛弃 (temp=1.2) 预设的三段格式，输出走向自由化。
  - **主题漂移**: 内容核心从输入相关的主要内容发散到各种相关性逐渐减弱甚至完全无关的内容。如输入程序员、零食补充相关，输出却含有“精神熵增”(temp=1.1) 或“午休效率提升”（temp=1.2）等内容
  - **语言俏皮**: 语言风格转向趣味化，出现“优秀代码 =...+ 能量管理 + 咖啡因 +...”的公式（temp=1.1），抑或“午休能量黑客” (temp = 1.2) 的口号等
  - **颠覆创意**: 模型产出不再是内容大纲，而是全新的概念或方案，如“程序员效率挑战赛”(temp=1.2) 等，虽偏离了原始任务，但确是一个极具价值的新营销案。

  结论:

  高温区间属于**创意风暴**，其产出结果较为新颖、视角更为独特，或许能带来意想不到的惊喜，也有可能是天马行空的想象（但难以执行）。其结果具有较大的跳跃性和不确定性，适合于**项目初期 BrainStorm 或项目中后期创意枯竭时**使用，以求打破思维定势，探索全新的可能性。

最终总结:

作为链接 Agent 1 和 Agent 3 的内部黑箱，我们在多次尝试不同温度的输出结果后，最终选用 0.75 的温度，在优先保证产出内容可以直接使用并投喂 Agent 3 以避免差之毫厘、谬之千里的前提下，对模型创新创意潜能激发以求达到一个中等的程度，以保证本部分的稳定型和一定程度上的创新性

5. 反思

5.1. 已有文档的阅读以及对大模型的应用

在项目最开始的阶段，由于并不熟悉 camel-ai 框架，我们花费了很长时间用于技术文档和 camel-ai 的 github 仓库的阅读。在发现靠人力阅读和查找问题并不现实后，我们开始尝试使用 deepwiki, 一个可以扫描指定 github 仓库来回答问题的大模型进行问题的询问，并尝试让其给出回答。
例如刘英哲的 deepwiki 使用记录: "https://deepwiki.com/search/deepseekchat-apimodel_9586235e-fdad-411b-8bc7-c042b49fc398"中，在尝试使用 WorkForce 时收到报错，但不清楚报错是如何产生的。在询问 deepwiki 后得到了解答。
同时 deepwiki 和 deepseek 等大模型也可用于生成一些简单的代码。例如敏感词检查部分的 AC 自动机代码就是由 deepseek 生成的。

由此可见，使用大模型辅助代码阅读和设计可以显著提升效率，在之后的学习和工作中也要善用之。

5.2. 关于团队协作

由于团队成员时间难以协调，且只有一位同学能正常通过命令行与 github 仓库交互，我们采取的是将整个代理分为三个部分，一人负责一个部分以在前中期避免需要反复更新代码（尤其是我们无法使用 github 进行快速更新的情况下)。这也带来了一个后果，也就是我们前期的交流不够充分的情况下，会出现代码重复甚至冲突的情况，例如代理之间文件内容的交接。这在后期浪费了我们相当长的时间用于 debug。因此在合适的条件下还是应当尽可能频繁地更新代码和交换意见，以避免冲突的发生。