**auto-coder.chat： 如何自动化开发基于大模型的应用**

我们将当前应用开发分成两种方式：

1. 传统应用开发，比如 web, APP，小程序，桌面软件，嵌入式软件等等。
2. 大模型应用开发，任何基于大模型驱动的应用，形态可以上述传统形态中的任何一种。

两者在编程范式上有一定的区别：大模型应用开发大量使用 prompt 和大模型做交互，并且输出有一定的不确定性。

auto-coder.chat 在传统应用开发上的能力毋庸置疑。那他用来开发当前流行的基于大模型的应用效果怎么样？为了解决使用大模型开发当前流行的基于大模型的应用，需要解决两个问题：

1. 功能需求自动转化成 合适的 prompt 和工程代码
2. 大模型需要能够熟练使用和大模型的编程交互的工具以及AI存储能力

第一个核心还是靠大模型自身的能力，但是计算大模型能产生好的prompt，但是还要解决两个问题：

一个是prompt的产生上下文准备。如果你用大模型去写 prompt ，你需要收集一些额外的信息，我在原始需求里，会让大模型参考这个参考那个去写prompt，你要是手动复制黏贴把这些需要大模型参考的地方整理成一个文档给到大模型也是很费劲的，而我这里只需要提下就行，auto-coder.chat 就自动给你收集好了，这个和传统编码是一样的，你要完成一个功能需求，就需要auto-coder.chat 给你收集上线文，否则他是没办法迭代项目的。

其次是写好的prompt你接着要改成代码里能用的东西，这也是要费很大功夫，你要把prompt模版化，哪些要变成变量，哪些要重新拼接，然后重新渲染,这也是很多大模型应用开发框架在解决的问题。像langchain/llama\_index之类的框架，大模型基本上无法利用他们的API 完成较为复杂的应用，这主要是：

1. 他们的API过于繁杂，导致大模型难以准确掌握或者通过阅读文档来掌握。
2. 他们的API因为早先设计的问题，导致都普遍进行了大量重构和更新，导致大模型以前学的知识都过时了。

整个大模型应用的基石有两点：

1. 如何和大模型完成交互来完成复杂的需求。
2. 如何高效的进行存储满足RAG类的需求。

我们为了能够让大模型方便的生成可以和大模型交互的以及完成RAG存储和查询代码，专门为大模型优化了一个库：

byzerllm(内置一个storage)，该库具有如下特点：

1. 接口足够简单但不是灵活性。接口简单体现在一页文档就能把和大模型所有的交互讲述清楚，灵活的地方是，这些交互又能满足几乎所有的需求。
2. 数据存储我们也提供良好的Python API ,可以让大模型更好的生成相关的操作代码，也是一页文档即可讲述清楚。

可以通过这一篇实例文章感受下我前面描述的事情：[让大模型使用 byzerllm 从 0 实现RAG应用](https://uelng8wukz.feishu.cn/wiki/KlHVwCilDi7y3LkSRblcAfqynpM)

在上面这个案例中，基于 auto-coder.chat 我们直接通过一个简单的”功能需求“生成了可用的应用，完成了向量化构建，查询，以及基于查询结果使用大模型进行回答的一整套流程。这个目前大模型使用非 byzerllm 库 还是很难做到的，其他的库都过于复杂，抽象层次过高，并且满足新需求接口变动频繁。

我们对召回的文本进行回答，那这里其实就需要一个 prompt, auto-coder.chat 不仅仅可以让大模型根据你的需求生成这个prompt,还是以函数的方式交付，你直接在程序调用这个prompt 函数就可以得到你想要的，至于如何和大模型交互，里面的prompt 真实是什么样子的，用户都可以不用关心，这就是 byzerllm 这个大模型编程接口的的魅力。

|  |
| --- |
| 很快，auto-coder.chat 也可以自动化完成包括大模型setup 的一些工作。因为 byzerllm 提供了 Python API 以及命令行API， 简单灵活且设计具有一致性，大模型很容易掌握。 |

下面 auto-coder.chat 自动生成的 prompt 函数：

|  |
| --- |
| Bash @byzerllm.prompt() def process\_query(context: str, query: str) -> str:  """  Based on the following context, please answer the query:   Context:  {{ context }}   Query: {{ query }}   Please provide a concise and accurate answer based on the given context.  """ |

你可以像下面这样调用这个函数，就完成了和大模型的交互，基本和一个正常的方法调用差别不大。

|  |
| --- |
| Bash context = results[0]["raw\_content"] response = process\_query.with\_llm(self.llm).run(  context=context, query=query\_text ) return response |