ChatGPT

A 股量化策略开发与演化体系设计

策略框架：适合 A 股市场的量化策略种类

A 股市场具备风格轮动快、主题炒作集中、小盘股高弹性的特征，因此应选择能捕捉这些特性的策略：

- **行业/风格轮动策略：** 根据市场风格周期快速切换投资方向的策略。例如根据政策风向或经济周期，在不同行业板块间轮动投资，以把握领涨板块的机会。由于 A 股存在显著的成长/价值、大盘/小盘风格轮动现象，如果能在恰当时机调整股票池，可获得较高收益 [1] 。这种策略适用于风格切换频繁的环境，需有指标跟踪行业相对强度、政策主题热度等来及时判断轮动信号。

- **多因子选股策略：** 综合多个因子（基本面、技术面等）筛选出优质股票组合，以获取稳定超额收益。在 A 股市场中，多因子模型已较成熟且有效 [1] 。常用因子包括价值（估值低）、成长、质量以及动量等。例如有实证表明"A 股小市值溢价"明显——小盘股长期回报显著高于大盘股 [2] ；同时"动量效应"在国内市场也有效，即近期涨幅大的股票短期内可能继续上涨 [3] 。因此多因子策略可结合小市值、高动量等因子筛选弹性大的股票，同时加入低估值、低波动等因子控制风险，以在风格转换中保持组合稳定 [3] [2] 。该策略适合中长期投资，通过定期调整因子权重来适应市场环境变化。

- **强势股追涨策略：** 聚焦短期内表现极强的股票（"龙头股"），在其上涨趋势中追击获取利润。A 股市场常出现**连续涨停**的强势股和集中的主题炒作现象，即所谓"妖股"或热点板块龙头。追涨策略利用市场对强势股的认可和资金惯性，如选取连续两天涨停且属于当前最热概念板块的股票作为次日买入对象 [4] 。为了控制风险，通常设置严格的买入/卖出条件，例如高开继续上涨时追入、达到止盈目标或日终平仓 [4] 。这一策略在牛市或单边上行行情下效果显著，适合小盘股高弹性行情。但在震荡市中连续涨停难以持续，容易"一日游"，需要结合**热点题材**筛选和止盈止损规则来避免追高被套 [5] 。

- **量价突破策略：** 基于技术分析的突破交易策略，在多周期观察下确认价格突破关键阻力位并伴随放量时介入。A 股投资者情绪容易集中，引发个股放量大涨。当某只股票在日线乃至周线级别形成向上突破且成交量显著放大时，往往意味着主升浪启动。多周期突破策略要求不同时间尺度上趋势共振，例如周线趋势向上、小周期出现放量突破买点，从而提高信号成功率。此策略利用小盘股高波动性的特征，及时抓住起涨点。同时，需要警惕假突破和交易限制（如涨跌停板制度）的影响，设置好止损以控制风险。

上述策略各有适用条件：轮动策略在风格变换明显时表现较好，多因子选股在长期投资和震荡市中较稳健，追涨和突破策略在单边上涨或主题行情中收益高但风险也较大。在实务中可以将多种策略组合，以适应不同市场阶段。

开发流程：从策略设计到回测、实盘的闭环

基于"韬睿量化系统"整体框架，构建从策略构思到实盘交易的**快速迭代闭环**工作流程：

1. **策略设计与定义：** 量化研究员提出交易策略想法，明确交易规则和逻辑。例如确定选股因子、买卖信号触发条件、风控规则等。充分利用A股的特性调整策略参数（如轮动频率、止盈止损比例）。这一阶段产出策略方案文档。

2. **策略代码生成（利用 Cursor + GPT）：** 在编程环境（如 Cursor AI 辅助编程工具）中，将策略方案转化为代码实现。使用 Cursor 集成的 GPT-4 助手，根据策略逻辑自动生成策略代码框架（Python 为主，兼容聚宽 JoinQuant API）。开发者可以通过自然语言描述模块需求，让 GPT 生成相应模块代码，再由人工检查完善。**Cursor** 支持调用GPT插件和指令，可将策略拆分为数据获取、选股、仓位管理、交易执行等模块分别生成，实现代码结构的模块化和清晰组织 [6] [7]。生成过程中注意与文件系统协作：Cursor 自动读取项目上下文，避免重复生成已存在的文件或函数，实现增量式开发。

3. **历史回测验证（基于聚宽等平台）：** 将生成的策略代码在**聚宽（JoinQuant）**等回测平台上运行历史数据检验。聚宽等平台提供了完备的历史行情数据和API，可以模拟策略在过去市场的表现 [8]。通过多周期、多市场环境的回测，评估策略收益曲线、最大回撤、胜率等指标。如有不足，返回调整策略逻辑或参数。利用 Cursor 对代码进行修改优化，GPT 助手可根据回测结果建议代码改进（例如调仓频率、因子阈值调整等），迅速迭代。这个设计-生成-回测的循环可多次执行，直到策略在历史数据上表现满足预期。

4. **代码部署与实盘连接（PTrade 执行）：** 当策略经过回测验证后，将策略代码部署到**PTrade专业交易系统**或券商量化交易平台上，实现实盘交易。PTrade 是高性能的量化交易执行系统，支持多市场品种实时下单和策略托管 [9] [10]。通过将策略代码按 PTrade 接口要求进行适配（例如采用其API封装策略逻辑），上传至 PTrade 云端运行。PTrade 提供低延迟的行情推送和交易执行环境，以及托管式服务，适合希望策略稳定 7x24 运行且不需人工盯盘的场景 [10]。部署完成后，可在模拟盘先行验证，然后连接实盘账户交易。

5. **实时监控与迭代优化：** 策略上线实盘后，需要实时监控其运行状态和绩效指标。通过韬睿量化系统的监控模块或 PTrade 自带的风控监控，跟踪每日交易信号、盈亏及风险暴露情况。如果出现异常（如策略失效或市场环境突变），及时暂停策略并分析原因。利用 GPT 助手和 Cursor，对新出现的市场情况进行分析，快速产出策略调整方案并生成更新的代码版本。然后再次在历史数据和实时模拟上验证。这个过程中还包括定期的策略评估会（例如每月一次），综合回测、实盘统计数据来微调策略参数乃至更换策略类型，实现策略的持续迭代演化。

整个流程形成从**想法→代码→回测→实盘→反馈**的闭环。借助 AI 辅助开发和一体化量化平台，显著缩短了策略开发周期，提升协同效率。在此体系下，策略研究员、AI助手与量化平台紧密协作，能快速将交易想法转化为实际收益。

Cursor 使用规范：高效利用 GPT 生成策略代码

在 Cursor 编程环境中利用 GPT 能大幅提升策略开发效率。为避免重复生成和确保代码质量，需要遵循一些使用规范：

- **充分利用 `.cursorrules` 进行项目约束：** 在项目根目录编写 `.cursorrules` 配置文件，向 GPT 提供策略项目的背景、模块划分、编码规范等信息 [6]。例如，指定"本项目为 A 股量化策略，包括数据获取、选股模型、交易执行三大模块"，以及代码需符合聚宽 API 规范等。通过将策略拆解为多个模块并为每个模块编写清晰的生成提示词，Cursor 中的 GPT 将更准确地按照要求产出代码片段 [6]。这种规则文件相当于为 AI 设定开发指南，能提高代码一致性、减少反复修改。

- **调用插件与工具：** Cursor 支持集成一些插件或命令调用。例如可以调用 **终端插件**运行生成的代码片段来检验语法，或调用**单元测试插件**对关键函数进行测试。合理使用这些插件，可让 GPT 在生成代码后自动执行简单测试，从而及时发现并修复错误。对于获取数据、绘图等需求，也可调用相应库接口，让 GPT 根据文

档编写调用代码。注意在 `.cursorrules` 中注明可用的库或插件名称，帮助 GPT 了解可以利用的工具（如指定"首选使用 pandas 处理数据"）。这样 GPT 会按照约定的库和插件编写集成代码。

- **代码结构模块化：** 将复杂策略拆分为若干文件或函数模块，GPT 生成时逐个模块完成。比如定义 `data_fetch.py`（行情数据获取）、`strategy_logic.py`（信号逻辑）、`risk_control.py`（风控管理）等文件。先让 GPT 生成每个模块的框架代码，然后再细化实现细节。模块化设计一方面使代码更清晰，另一方面减少了 GPT 每次需要考虑的上下文范围，从而减少重复生成整个策略的情况。QMT 等专业量化平台也提倡策略**模块化设计**，将复杂策略拆分优化后再组合[7]。在 Cursor 中，可通过逐步对各模块对话的方式，让 GPT 专注于当前模块的编写或修改，不去改动其他无关部分，避免反复重写已有代码。

- **避免重复调用同一功能：** 利用 Cursor 的上下文记忆和**规则管理**来防止 GPT 多次输出相同内容。Cursor 会索引项目中文件内容，可在对话中让 GPT "参考已有的函数实现不要重复定义"。另外，通过 `.cursorignore` 文件忽略掉不必要的大文件或已稳定模块，防止它们干扰 GPT 对当前任务的专注[11]。例如已生成的辅助函数文件可暂时加入忽略列表，则 GPT 不会再试图改动它们或重新生成。开发者也应在对话提示中明确指出哪些部分需要新生成，哪些部分保持不变。这种约束能减少 AI 无效输出，提升调用效率。

- **项目结构优化与版本控制：** 随着策略开发推进，要保持项目文件结构清晰，并使用版本控制系统（如 Git）管理每次重大修改。在 Cursor 中编写代码时，可以定期保存当前版本，由GPT在提交前总结变更要点。这既可以为后续对话提供参考（GPT可被告知"与上一版本相比，我修改了止损逻辑"来避免重复解释），也方便人工回溯。借助 Git 等版本控制还能避免 Cursor 再次生成旧版本里已经存在的代码——因为通过对比差异，开发者知道哪些是新增需求，进而在提示 GPT 时**只聚焦新需求**部分，减少重复劳动。

- **结合文件系统协同：** Cursor 允许 AI 读取和写入项目文件。因此可以让 GPT 在生成新代码前**先检索项目中是否已有类似文件或函数**。例如在让 GPT 编写一个名为"strategy.py"的策略文件前，先要求它检查项目目录避免命名冲突或重复。对于已有文件，让 GPT 进行"追加修改"而非重新生成整个文件内容，从而防止重复片段出现。如果必须生成新文件，建议采用有序的命名（如strategy_v2.py），并删除或存档旧版本，以避免多个类似文件共存造成混乱。通过合理引导，GPT 可以像团队协作一样与文件系统互动：读入现有代码->基于新需求生成补充代码->保存，这一过程确保不产生重复冗余的文件。

总之，在 Cursor 中使用 GPT，需要制定**清晰的规则与上下文**让 AI 明白项目全貌，尽量让它增量式地编写代码。配合适当的插件测试和人工review，能够实现高效的策略代码生产，避免重复造轮子和代码冲突。实践证明，精心维护 `.cursorrules` 和 `.cursorignore` 并定期更新，可使项目始终整洁高效[12]。

策略版本数据库与规则系统：避免重复输出

为防止 GPT 反复生成已做过的策略，建议构建一个**策略版本数据库**及配套规则系统来记录每次策略生成和修改的历史。其设计要点如下：

- **版本记录数据库：** 采用 SQLite 轻量级数据库或类似文件索引系统，建立表结构记录策略版本信息。字段包括策略名称/编号、版本号、生成时间、策略描述摘要、主要参数、以及对应的代码文件存储路径等。每当通过 GPT 完成一次新策略生成或重大更新时，将相关信息插入数据库。这样可以方便查询某一策略是否已存在或上次修改是什么时候。

- **策略指纹与重复检测：** 可以为每份策略代码生成指纹（例如对核心逻辑文本计算哈希值）存入数据库。当用户提出一个策略想法请求 GPT 生成代码时，先检索数据库中**是否有相同或相似的策略**。相似度判断可基于策略描述的关键词匹配，或利用向量检索（将策略描述嵌入为向量，与历史描述比较）。如果发现已有**高度相似**的策略版本，系统即可提醒用户并提供历史版本内容，而非让 GPT 再次输出相同内容。这套规则有效避免GPT对同样需求重复劳动。同时也保护既有成果不被新的对话覆写。

- **版本迭代规则：** 当用户确实需要在已有策略基础上微调时，可以触发**迭代模式**：系统提取旧版本代码片段，提供给 GPT 仅让其在此基础上改进特定部分。例如用户提示"在现有动量选股策略上增加市值因子过滤"，则将之前版本的代码和修改要求一并交给 GPT，生成的新代码即视为下一版本。在保存时赋予新版本号并归档旧版本。这种规则管理确保 GPT 明确知道是在做增量修改，输出不会与之前完全重复 [5]（这一引用侧重说明策略需要在不同市场状态下调整，而我们的系统通过版本管理来实现有依据的调整）。

- **文件系统配合：** 策略版本文件可以按**版本号或日期**命名存储，例如 `strategy_X_v1.py`，`strategy_X_v2.py` 等，并在数据库中记录对应关系。这样当用户查询或GPT检索时，可以直接定位最新版本文件进行读取或更新。如果采用单一文件迭代，也应将旧版本保存副本，以防出现GPT误改无法找回的情况。规则系统可以规定**同一策略**的输出频率，如非重大市场变化，不允许 GPT 频繁地产生内容相同的新版本，避免浪费资源。

- **辅助记忆和提示：** 利用上述数据库，系统还能对 GPT 提供**额外上下文**："策略X上一版本的收益率为10%，本次希望优化风险控制部分"。GPT 在生成时就能结合这些记忆，只针对需要改进的部分输出新内容，从而不会重复整段策略描述或代码。长远看，可将策略版本库与GPT的**长期记忆模块**相连，使 AI 对已做过的策略有所"印象"，提高对重复请求的判断力。

通过数据库记录和规则约束，我们建立了一套**策略知识库**，不仅避免重复输出，也沉淀了策略演进过程的宝贵数据。量化团队可以随时查询过往版本绩效和修改理由，结合GPT的分析生成报告，判断下一步优化方向。这实现了策略开发的**知识闭环**：历史经验反馈入新策略设计，防止无效重复工作。

MCP 工具链推荐：多智能体协同与记忆管理

为实现上述协同开发与策略更新流程，推荐采用一款**多智能体协作（MCP）工具链**来 orchestrate 各模块。综合考虑功能完善性与社区成熟度，这里推荐使用 **LangChain** 框架（以及其扩展组件）作为构建该系统的底层MCP工具。

为什么选择 LangChain： LangChain 是当前流行的开源框架，擅长将大型语言模型与各种工具、记忆模块结合，便于构建复杂的链式任务流程 [13]。针对我们的需求，LangChain 具备以下优势：

- **管理多任务与多智能体：** 可以为每个策略开发任务创建独立的链或 Agent。比如同时开发多个策略时，LangChain允许并行运行不同的链，每条链维护自己的对话状态和工具调用，不会互相混淆。还可创建一个协调代理（Controller Agent）来管理多个子代理，各司其职（如一个Agent负责检查版本库，另一个负责调用GPT生成代码），实现多 Agent 分工协作。这种多智能体工作流在 LangChain 中可以通过 AgentExecutor 等组件配置实现，也可借鉴微软的 AutoGen 框架来加强复杂协作。 [14]

- **调用 GPT 模型生成代码：** LangChain对接了OpenAI的GPT-4接口，使用其 LLM 模块可以方便地调用 GPT 生成所需内容。开发者可以编写 PromptTemplate 来定义生成策略代码的提示，并通过链逻辑将用户输入、历史版本信息等拼接成提示词喂给 GPT。LangChain 还支持 **函数式调用**和**插件Tool调用**，这意味着

GPT 可以在链中自动使用我们定义的工具（如"回测工具""版本查询工具"）完成特定任务。当需要生成代码时，它调用OpenAI LLM; 当需要运行回测时，可以触发预先集成的回测函数。这种灵活的工具调用机制使整个流程高度自动化。

- **记忆维护与判断重复：** LangChain 提供了多种记忆（Memory）机制，可用于让 Agent 保留先前对话和重要信息。比如使用 **ConversationBufferMemory** 可让 Agent 在连续对话中"记住"已生成的代码摘要，不会再次询问已经确定的内容。此外，针对长期策略版本记忆，可以将 SQLite 数据库或向量存储集成为 LangChain 的记忆后端。更先进的用法是引入**LangChain扩展 LangGraph**，它支持 Agent **持久化记忆**，允许智能体记住过去的交互并利用这些信息指导未来决策 [15]。借助这些记忆功能，我们的MCP系统可以在每次生成策略前查询内部知识：如果发现与过去记录相似，就判断无需重新生成完整代码，只需微调。这等于给智能体增加了"经验判断"能力。例如LangGraph引入了状态图，让Agent具备循环调用LLM和自我检查的能力，可反复确认是否真的需要新代码 [15]。因此，LangChain系工具能有效辅助决定"重新生成或微调"。

- **决策逻辑与规则实现：** 利用 LangChain，我们可以在 Chain 中嵌入自定义的逻辑判断节点。例如在调用GPT生成代码之前，先调用一个Python函数查询策略版本库（这本身可作为一个Tool）。函数返回一个标志指示是否已有类似版本。如果是，则链路可以**跳过**GPT生成步骤，直接调取历史代码或让GPT基于历史版本给出改进建议。LangChain 的灵活之处在于我们能构建这样的**条件链**：根据工具输出选择下一步走向。这实现了"判断是否需要重新生成代码"的自动化。如果只需细微调整，Chain 可以直接调用另一个 PromptTemplate，提示GPT"请基于以下旧代码做小幅修改"而不是从头生成。整个流程在LangChain框架下可维护得清晰可控。

- **易于集成外部组件：** LangChain 兼容诸如 Flowise 等上层工具，也支持与 AutoGen Studio 等多智能体UI配合。如果团队希望可视化地管理这些链和Agent，也可以将LangChain逻辑接入 Flowise，用拖拽方式管理策略任务流程 [13]。LangChain本身有丰富的社区支持和文档，易于查找实例（例如很多关于**Task chaining**和**Agent工具**的范例）。此外，它能无缝调用Python环境函数，这意味着我们先前提到的聚宽回测、PTrade下单等都能通过封装为LangChain的自定义Tool来实现。一旦GPT产出代码，系统甚至可以自动调用回测Tool跑一下结果再将摘要反馈给GPT做二次分析，形成闭环优化。

综上，采用 LangChain 构建的 MCP 系统可以胜任管理多策略开发任务、调用 GPT 生成策略、维护对话和数据库记忆，并通过灵活的链式逻辑判断何时需要新的代码生成或简单调整。这样的平台将各个环节打通：**想法产生→AI生成→结果评估→智能决策下一步**，真正实现策略开发的**高效、模块化、智能化**。当然，市面上还有其他选择，例如 **AutoGen Studio** 提供了无代码界面便于多 Agent 工作流设计，或 **Smol Developer** 专注于整个代码库的自动生成 [16] [17]。这些也各有优势：AutoGen Studio让复杂代理协作更直观，Smol Developer能够根据规范一步到位产出多文件代码并处理跨文件依赖 [18]。团队可根据偏好选型。但综合考虑可定制性和社区成熟度，LangChain 是较为稳健的方案，并可在此基础上引入上述工具形成混合链路。

结语： 借助以上体系，我们能够搭建起一套适用于 A 股市场的**高效、模块化、智能演化**的量化策略开发流程。从策略构思，到 AI 自动生成代码和快速回测验证，再到实盘部署和自适应优化，每一步都有协同机制和工具支持。这套闭环体系将大大加速量化策略从想法走向交易盈利的过程，帮助团队在瞬息万变的 A 股市场中保持策略的竞争力和前瞻性。通过人机协同和流程自动化，量化投资进入"快速迭代"的新阶段。

参考文献：

1. 李奥辉, 刘胜题. 基于*A*股价量因子的行业轮动策略研究. 运筹与模糊学, 2023 [1]

2. BigQuant量化策略分享. 动量多因子选股策略 3 2
3. BigQuant量化策略分享. 昨日涨停打板策略 4 5
4. 木头量化. 量化交易系统*QMT*与*PTrade*的区别、优势 8 10 7
5. CSDN博客. 集成*GPT-4*的代码生成器*Cursor*使用体验 6 12
6. AITNTNEWS人工智能. 智能体工作流开源项目大盘点 13 15
7. Metaschool. *Smol Developer: AI-powered code generation framework* 16 17

---

1 tipdm.org
https://tipdm.org/u/cms/www/201908/081056569at4.pdf

2 3 策略分享-动量多因子选股策略 - BigQuant量化交易
https://bigquant.com/wiki/doc/YSonXtEzFI

4 5 昨日涨停打板策略 - BigQuant量化交易
https://bigquant.com/wiki/doc/fx0Bpf9XM9

6 11 12 集成 GPT-4 的代码生成器 Cursor 使用体验如何？怎么用更高效？_cursor 集成.cursorrule-CSDN博客
https://blog.csdn.net/hellocode_/article/details/144936303

7 8 9 10 量化交易系统QMT与PTrade的区别、优势、量化交易策略， - 木头量化入门 - 博客园
https://www.cnblogs.com/bigleft/p/18286458

13 14 15 智能体工作流开源项目大盘点，20个项目轻松构建Agentic Workflow
https://m.aitntnews.com/newDetail.html?newId=6203

16 17 18 Smol Developer: Features, Use Cases & Alternatives
https://metaschool.so/ai-agents/smol-developer