今天派活有点少，除了上了个很小的快修之外就只有协助运营和测试干一些小杂活小支持了。

之外的整个时间大头是在做一些小训练，小头是 review 一下组里别人的代码。

fix 没啥好说的，关于 social-fantasy-be/pull/34，支撑 QPS 的核心还是寄希望于成熟的缓存策略。

另外，这种多匹配策略按各自占比共同提供候选页的好处是可以平衡日常运营和新策略实验，因为权重占比低的新策略哪怕收敛速度会降低（如果需要反馈学习的话），但对用户体验的影响会小。

写太短了就贴一下今天自己做的训练凑个字数吧。

14:39: 好吧，思考了一下觉得 Dispacher 的模式不怎么样。干脆实现一个消息总线吧，或者调研一下有没有现成的。实际上之后悬浮求组件要扩展成与本地存储沟通的桥梁，也会是个比较大型的东西；另外，core-runner 也会与 UI 层有频繁的沟通。我在想，说是 core 层和 UI 层有双向的消息交互，但实际上核心的点还是 core 层为主导，从 UI 层到 core 层的消息一般不会引起 UI->core->UI->core… 这样的深层传递，最多只会有 UI->core->UI 这样三层的消息同步，而且应当适当为 UI 层加入锁定机制以免状态机混乱。

15:47: 其实状态共享层也有些多余，因为 core-runner 层真正需要状态共享的数据是易失的，比如虚拟文本块的输出内容和虚拟文本块是否冻结（实际上当用户想要保存工作流的时候软件会建议用户把冻结内容的虚拟文本块转换成普通文本块），而如今 core-runner 层需要交互的底层组件就只有 db 了，根本不需要共享状态。相当于 IR 只出现在有多种架构和多种系统的时候，但你只有一种架构和一种系统，那么 IR 就不必要了。

> 500 {“error”:{“type”:“new_api_error”,“message”:“当前模型 claude-opus-4-5-20251101 负载已经达到上限，请稍后重试（request id: 20260119165227662265918pWLeWCcA)”},“type”:“error”}

16:27: 公益站的 Opus 被佬友蹬爆了，先来想想 core 组件的重构。嘛，最重要的设计原则就是分离 metadata 和 running-state，如 running | pending | failture 和 freeze: true | false 这些都应该放到 core-runner 层里作为 running-state，而 core 层的组件定义应该只涵盖 metadata 以保证简洁性。