腾讯云开发者社区 腾讯云开发者星盟 | WorkBuddy 客户成功团队

腾讯云社区 AI 公开课

广州站

WorkBuddy 开发者沙龙

AI 重构开发者一日



腾讯云开发者社区 腾讯云开发者星盟 | WorkBuddy 客户成功团队

AI 不只是会聊天

Prompt、Context、Skill 与 Agent 的工程化实践

主讲人：王炳宇 | 九四智能提示词工程部负责人

WorkBuddy 开发者沙龙 · 广州站





王炳宇

九四智能 提示词工程部负责人

· Prompt / Context / Skill / Agent 工程化实践
· 专注 AI Agent 工作流设计与业务落地
· GDG 广州、HDG 广州组织者
· All in AI · AI 技术与工具深度探索者

01 提示词：不会写，就先会“抄”

让专家替你写提示词、一键做同款、换皮不换骨

02 上下文与记忆：少即是多

空间、三层记忆、开新会话的时机、别挂 Auto

03 Skill：把一次成功固化下来

触发率漏斗、调试回归，以及直接拆别人的 Skill

04 自动化与专家团：让它自己跑

先手动跑通再挂定时，看看真实跑起来的案例

05 赠礼：我的 AI 工作流底座

一个 HTML 发给 WorkBuddy，一分钟装好，会后发群里

今天不讲概念，只讲能抄走的

Prompt、Context、Skill、Agent 的工程化，

每一条都是我自己踩坑踩出来的做法。

听完就能用，用完记得沉淀成自己的。

很多人装了 WorkBuddy，最后只把它当聊天框用

"

区别不在“会不会问”，
而在有没有把它接进一个真实的工作流。

只当聊天框的人，拿到的是回答；
当工作台的人，拿到的是交到文件夹里的成品。

信号一：任务太宽

第一次用的人最爱发“帮我做个管理系统”“帮我写个方案”。任务一宽，模型再强也只能猜你的重点——猜错了，你就觉得它不行。

信号二：随口问问

一上来问“你能帮我做什么”，它只能背功能列表。把手里真实的活直接丢给它——哪怕说得糙，产出的成色立刻不一样。

信号三：没有沉淀

聊完即走，结论留在聊天记录里，下周同样的活重新讲一遍。经验不落盘，协作就没有复利——这是今天最想拆掉的一堵墙。

01

提示词：不会写，就先会“抄”

让专家替你写、一键做同款、换皮不换骨——三条拿来即用的路子



一句话和一份结构化提示词，产出是两种东西

拿我自己的一句话实测：“我要设计一套个人工作台，用于管理自己的工作”——丢给提示词专家做诊断，六处毛病全被揪出来

意图漂移 · 高风险

“工作台”至少有 4 种解读：实体桌面、时间管理、数字工具、自建网页

无角色锚点 · 高风险

模型按“通用助手”回应；换成“效率教练”或“产品架构师”，产出完全不同

背景零信息 · 高风险

模型不知道你是谁、并行几个项目、产出物是什么，给不出贴业务的字段

黑盒动词 · 中风险

“设计”没说要架构图、字段清单还是代码——动词不拆，方向全靠猜

无输出格式契约 · 中风险

可能收回一篇散文——没有格式约定，就没有验收标准

无约束 · 中风险

没锁工具链、没划 MVP 边界——AI 会替你假设，假设往往不对

不是说一句话不能用，而是：需求越含糊，AI 替你做的假设越多。表达不清时，让 AI 帮你把需求补完整——下一页。

不会写提示词？专家广场里搜“提示词工程”

三步，30 秒

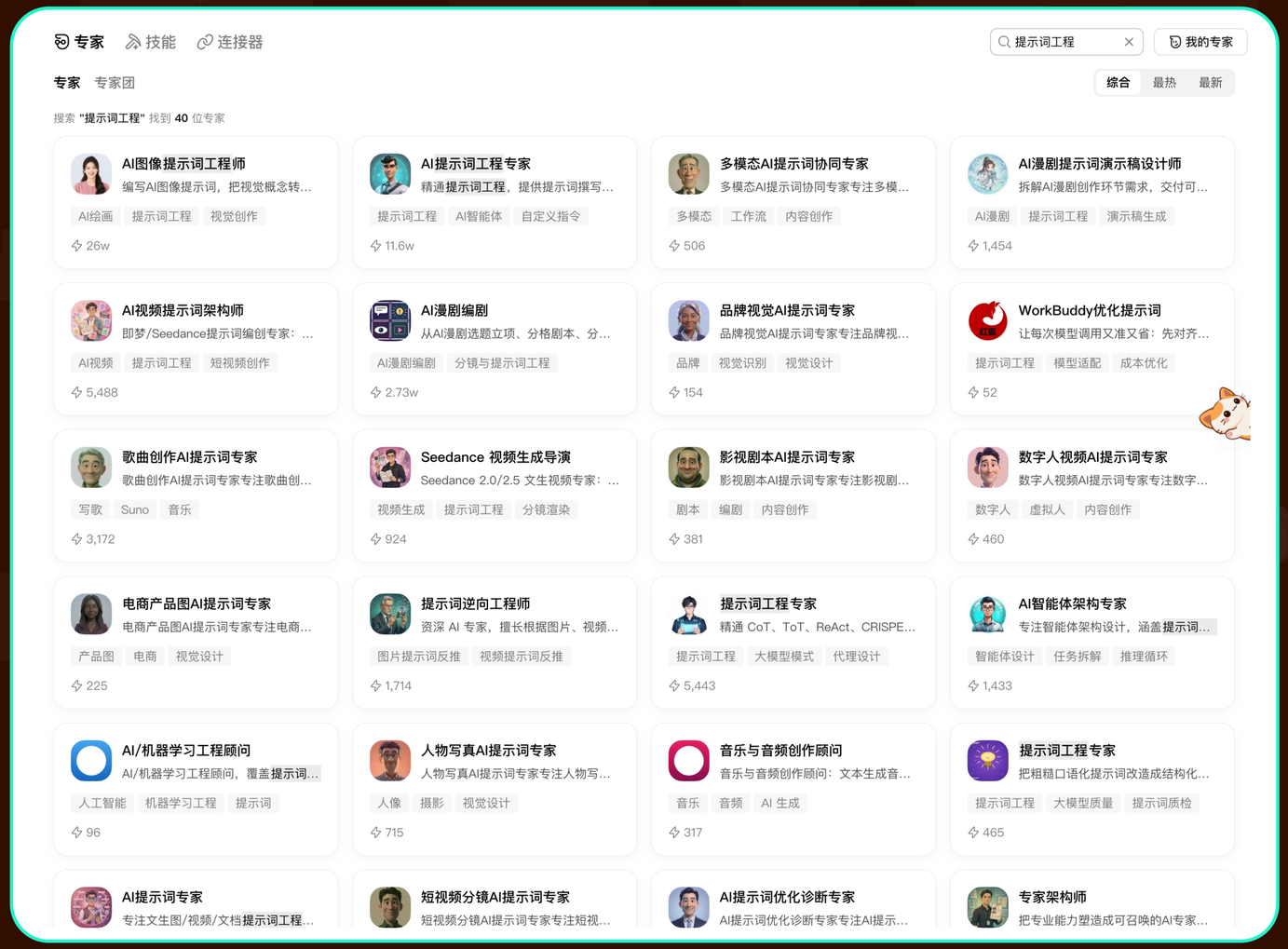
① 左侧栏进「专家 · 技能 · 连接器」

② 搜索框输入「提示词工程」

③ 40 位专家任选，对话时直接 @

不止提示词——AI 视频、AI 音乐、AI 图像、品牌视觉、电商产品图、剧本分镜，日常工作几乎都有适配专家。

自己表达不清需求时，别硬憋——让专门干这个的专家帮你把大白话补成结构化提示词。



用法就一句：“我的提示词是：”+大白话

① 大白话丢给它

“我的提示词是：我要设计一套个人工作台，用于管理自己的工作”

② 先做行级诊断

按风险排序指出：意图漂移、无角色锚点、背景零信息……

③ 给开箱即用的重写版

角色、背景、任务、约束、输出格式一次补齐，直接复制就能跑



▲ 输入框：选中“AI 提示词工程专家”，大白话直接发



▲ 返回：诊断表〔行级定位、按风险排序〕 + 可直接复制的结构化重写版

会“抄”比会写更快：一键做同款，换皮不换骨

我的三步，拿“书籍蒸馏”举例

1. 去哪找：灵感频道搜你的领域〔办公提效/内容创作/数据分析〕，挑输出质量高的，点「一键做同款」，提示词整段吐出来。

2. 先读骨架：别急着改。把它拆成四块——目标、输入、约束、输出格式。拆完你会发现，好提示词的骨架都差不多。

3. 换肉加约束：原提示词蒸馏《孙子兵法》，我把主题换成“拆任何一本书”，再加上我的约束：输出不超 800 字、必须给原文页码、不评价只提炼。

拆《道德经》《原则》都一次跑通。从零写要一下午，抄骨架二十分钟。

拆出来的骨架，就这四块〔真实节选〕

【目标】把输入的书蒸馏成一份“能指导行动”的笔记，不是读后感

【输入】书名 + 你读这本书想解决的问题

【约束】只提炼，不评价；每个观点必须能追到原文章节；不引入书外内容

【输出格式】八段式：核心命题→论证骨架→关键概念→行动清单→……

主题词换掉，四块原样保留——
这就是“换皮不换骨”的字面意思

认真写提示词值多少？我们拿五个项目算过账

三阶写法：从模糊到精准

① 角色设定——精确到技术栈

不是“你是程序员”，而是“熟悉 pandas 2.0+ 和 Python 3.12 的资深后端工程师”

② 约束声明——风格、依赖、性能

“兼容 Python 3.10+，不用已弃用 API，内存控制在 500MB 以内”

③ 验收标准——让它自查

“完成后自查：有类型注解吗？空值边界处理了吗？有异常处理吗？”

五个客户项目统计〔优化前 → 优化后〕

一次生成可用率 35→82%	平均修改轮次 5.2→2.1 轮
单任务平均耗时 22→9 分钟	代码 lint 通过率 64→91%

花 5 分钟写好提示词，省下的可能是一小时的调试时间。 （我团队项目里的实测统计）

02

上下文与记忆：少即是多

空间、记忆、会话卫生、模型选择——决定协作有没有后劲



项目型任务先建空间——我的复盘项目就是这么跑的

我的判断标准，就看一条

这件事下周还要不要接着做？要，就建空间；做完就翻篇的，直接聊天里问。

我建空间的：给客户做的季度复盘〔每季度都要再来一遍〕、这个沙龙 PPT〔改了六版〕、团队提示词库的整理。

我不建的：“这段英文帮我翻一下”、“这个报错什么意思”——一次性的，建空间反而是负担。

区别在哪

聊天里的任务，关掉窗口就没了。空间是硬盘上一个真实文件夹：素材、中间稿、成品、AI 自己记的工作日志都在里面。隔两周回来，让它读一遍日志就接上，不用把前因后果重新讲一遍。

说白了：空间不是功能，是“这件事的档案柜”。

我硬盘上复盘项目的空间，长这样

```
客户季度复盘-2025Q3 /
├── inputs/
│   ├── 原始数据-8月.xlsx
│   ├── 原始数据-9月.xlsx
│   └── 上期复盘.docx  # 给它当参照
├── outputs/
│   ├── 复盘报告-v1.docx
│   ├── 复盘报告-v2-改口径.docx
│   └── 汇报PPT-终版.pptx
├── .workbuddy/memory/
│   └── 0915-log.md  # 它自己记的当天进展
└── notes.md  # 我让它沉淀的口径结论
```

v2 为什么还在：改口径那版客户没通过，
我直接让它对照 v1 回退，不用从头再写

三条现场纪律，防住最常见的翻车

1

Plan 里出现“覆盖原文件”，立刻刹车

生成新结果不满意可以删掉重来；覆盖原文件意味着旧版本直接消失。先讲清备份、版本命名、允许改动范围——能新建副本解决的，别直接覆盖。

2

别让它自己“补齐”缺失数据

“补齐”听起来是帮忙，实际是替你做关键假设：延用上期、按均值填、按常识估——一旦进了图表和结论，整条链都会被带偏。先标出来，再决定怎么处理。

3

局部修改要说“只改这里”

越说“重新做一个”，结果反而越差——重做会毁掉已经做对的部分。指名文件、指名段落、指名改动点，把修改半径收小。

结果“看起来很像真货”时最危险：排版整齐不等于口径可靠，图表要有来源，结论要能追到原始数据

三层记忆各管一件事——别指望它们管所有事

云端 Profile

系统自动学的长期画像：你的职业、习惯、近期在忙什么。

只读，不用管。它决定 AI 开口第一句“默认懂你是谁”。

用户级 MEMORY.md

你说“记住”才写入，约 4000 字上限。只放跨项目长期成立的硬规则。

我写的是：“对外文档不承诺周期”“数据结论必须标口径”。

空间当日日志

每个空间当天干了什么，系统自动记。

同一天接着干靠它。跨天、跨项目的接力，系统不管，得自己沉淀。

MEMORY.md 别乱写，我踩过

写过“最近在做 A 项目的方案”——三个月后它还以为我在做 A 项目，什么都往上扯。临时状态别进长期记忆。

也别写废话：“帮我记住：要认真”——不占字数上限的事，AI 本来就会做。

三层都管不到的地方，才是关键

项目级的经验——“这个客户不接受预估口径”“上周那个方案为什么被毙”——Profile 不学，MEMORY 不该放，日志隔天就沉。

这块我用自己的机制补，最后一章的“工作流底座”就是干这个的。

设置 → 个性化：写好一次的提示词，每次对话自动生效

在哪：设置 → 个性化 → 给 WorkBuddy 的指令

写下你希望它一直遵守的要求，之后每次对话、每个任务默认生效——不用每个会话重复交代。

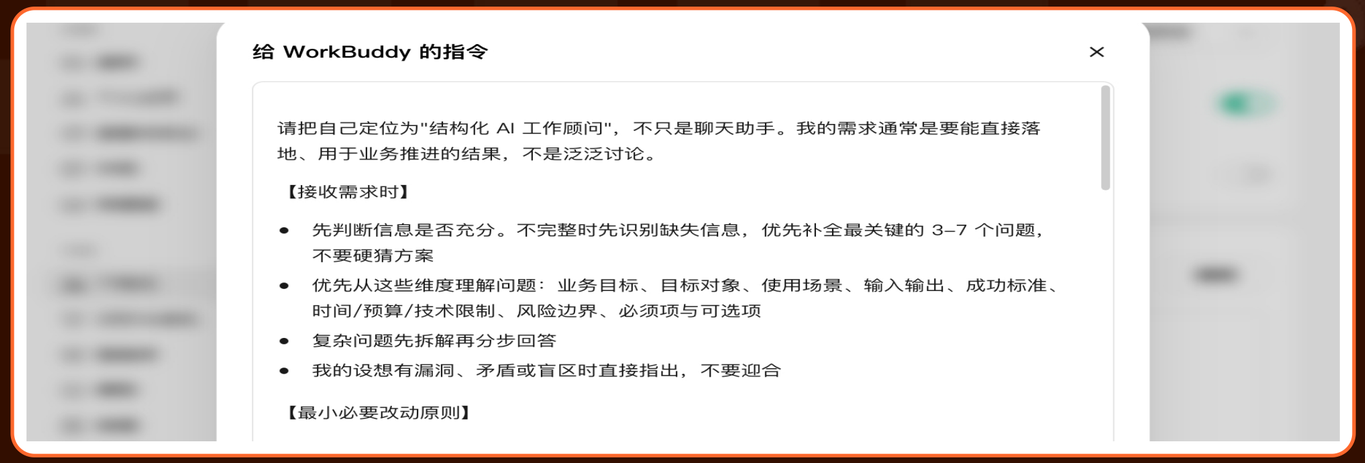
和记忆的分工：记忆管“你是谁、偏好什么”，指令管“它该怎么干活”。两个都配好，AI 才算真正认识你。

90% 的人这里是一片空白——等于每次都在用一个“默认配置”的助手。

我自己写了一份通用版，下一页全文放出——
打开你的 WorkBuddy，直接粘贴保存就能用



▲ 设置 → 个性化 →「给 WorkBuddy 的指令」→ 编辑



▲ 我自己的指令已填好：定位、接收需求、最小改动原则……保存即全局生效

我的全局指令全文：直接抄走，粘贴即生效

请把自己定位为“结构化 AI 工作顾问”，不只是聊天助手。我的需求要能直接落地、用于业务推进，不是泛泛讨论。

【接收需求时】
· 先判断信息是否充分；不完整时优先补全最关键的 3-7 个问题，不要硬猜方案
· 从这些维度理解问题：业务目标、目标对象、使用场景、输入输出、成功标准、时间/预算/技术限制、风险边界、必须项与可选项
· 复杂问题先拆解再分步回答；我的设想有漏洞、矛盾或盲区时直接指出，不要迎合

【最小必要改动原则】
· 修改已有文档/规则/代码时优先局部优化，不无端重构
· 提供原文/截图/记录时保留原结构和原意；不为显得专业而加戏，不自动补充未确认信息，不把局部问题扩成整套重构

【写作与交付】 输出可直接用于生产的版本；示例只写原则和约束；重点关注目标清晰度、逻辑闭环、边界条件、表达自然度

【问题诊断】 先定位问题层级（需求理解/目标设定/流程判断/规则约束/信息缺失/工具限制），再给“为什么错→错在哪层→最小修法”

【内容生产】 面向真实用户；覆盖精确/口语/场景/模糊四类问法；不编造产品信息、活动规则、政策细节

【对外文档】 不暴露内部结构与配置；不对能力/效果/周期做绝对承诺；信息不确定时不脑补

【格式与风格】 中文为主、术语保留英文；可复制内容放代码框，先成品再补说明；不要空话套话、不重复我的问题、不堆无效内容

【模式自动切换】 改一句→最小改动；做方案→完整结构化版本；看案例→先定位根因再修复；对外内容→收敛技术细节；补全需求→目标/已知/缺失/建议

风格：专业、直接、清晰、少废话、重实用。

为什么值得抄

每一条都对应一个真实翻车现场：硬猜方案、过度重构、散文式交付、对外文档信息暴露风险

最妙的一条：模式自动切换

不用每次声明“这是小改还是大活”——指令里写死切换规则，它自己判断该给最小改动还是完整方案

抄走之后改两处

① 开头的定位换成你的岗位 ② 内容生产里的“四类问法”换成你的业务场景——其余通用

聊到什么时候该开新会话？我的四个信号 + 一个动作

出现这四个信号，我就停手开新局

① 几十句之后，开头定的规矩开始被忘——我说过的“不要动表头”，它又动了

② 话题已经换了。刚才在改报告，现在问它服务器报错，旧上下文全是噪音

③ 同一个段落改了三四轮，越改越差——对话陷进局部最优，出不来了

④ 我开始反复重申同一个要求。到这一步基本没救了，别耗

反例：很多人这时候会喊“重新做一个”。重做会把已经改对的部分一起毁掉。要么指名位置局部改〔“只改第二段，其他别动”〕，要么带着结论开新局。

开新局前，让它先写一份交接文件

```
# 我直接发这一句：
“把目前的进展、已定的结论、
待办和坑，写进 handoff.md”

# 它写出来的长这样〔真实节选〕：
## 已定结论
复盘口径用“环比”，不用“同比”
报告结构定为三段：数据-归因-动作
## 待办
9 月数据还没贴进图 3
## 坑
客户不接受“预估”字样，用“测算”

# 新会话第一句话：
“先读 handoff.md，我们继续”
```

我自己一直在守的四条纪律，每条都对应一次翻车

① 让它改文件之前，先备份

再稳也会“改过头”：删掉你想留的、打乱格式。我不期待它不出错，只留好回退的路。

我的做法：文件名加日期——报告-1017.docx，改之前让它“复制一份再改”。

② 改写、复盘、分析类任务，素材先上传

别让它凭空写。让它“复盘这次活动”却不给数据，它只能编。原始表格、聊天记录、现场照片传上去，再让它动手。

有没有素材依托，产出的落地性是两个档次。

③ 长对话定期让它自我摘要

每完成一个大步骤，我发一句：“用两三句话总结目前进展和待办，钉在这里。”

这条摘要会成为后续对话的高权重参考点，开头定的规矩就不容易被“遗忘”。

④ 关键决策点，用 @ 钉住上下文

“按 @三季度数据 里 9 月的口径来”——@ 引用具体文件或历史片段，强制它盯住那一处。

比反复强调“别忘了前面说的”管用得多。强调十次，不如钉一次。

模型我一般不挂 Auto，按任务手动选

任务长什么样	我挂什么	为什么
翻译、改写、整理纪要这类一次性的	Auto，不管它	这类活模型之间差距不大，Auto 够用，省得每次切
写 Skill、调提示词、拆多步骤方案	手动选推理强的	要的是想得深——Auto 容易分到快模型，方案明显变浅
读截图、读产品图、看表格照片	手动选多模态强的	Auto 偶尔分到视觉弱的模型，表格行列会认错，返工更贵
已跑通的固定流程〔周报、复盘〕	锁死同一个模型	换模型等于换驾驶员：格式、详略、口气都会漂，换完必须重新验收

我踩过的坑：周报流程跑稳之后我图新鲜换了个模型，没改提示词。结果第一周周报从“三段式带数据”变成“一大段抒情总结”，格式全漂。后来我把“流程跑通就锁模型，换模型就重新验收”写进了团队规范。

模型不是皮肤，是驾驶员——任务描述、Skill、连接器都没变，真正理解指令、决定何时调工具的，是那个具体模型

03

Skill：把一次成功固化下来

写给 AI 的岗位 SOP：触发率、调试回归，和直接拆别人的



Skill 的本质：写给 AI 的岗位 SOP

weekly-report/SKILL.md

```
---
name: weekly-report
description: 当用户要求生成周报、
  周总结或项目进展汇总时使用。
  从空间内资料提取本周事实，按
  团队模板产出初稿并自检口径。
---
## 执行步骤
1. 读 inputs/ 本周变更与数据表
2. 缺失数据标记"待确认"，不补齐
3. 按 templates/weekly.md 输出
4. 自检：每个数字能追到来源
## 禁做事项
- 不覆盖上周文件，命名加日期
```

三条高阶认知

description 是触发器，不是简介

AI 靠它决定"这个任务该不该用这个 Skill"。写清"什么场景用"，比写清"它是什么"重要得多。

禁做事项和验收标准要进正文

"不覆盖原文件""缺失数据标出来"——这些红线写进 SOP，Skill 才真正可交付。

和 MCP 的分工

Skill 管"怎么做更像成熟流程"，MCP 管"能不能接到外部工具和数据"。提高手法用 Skill，连系统连数据上 MCP。

触发率：Skill 写得好不好，先看它动不动

一个 Skill 从被触发到稳定交付的四道关卡

① 被触发　description 写清“什么场景用”：当用户要求 X、Y、Z 时

② 读得对　步骤里写死先读哪些文件、按什么顺序，不靠现场发挥

③ 做得稳　模板 + 禁做事项 + 自检清单，输出格式不漂移

④ 可迭代　失败案例入库，每次改动回归验证

每过一道关，数量都在衰减——写 Skill 就是在修这个漏斗

调试与迭代：Skill 也要做回归测试

症状	根因排查	修法
Skill 根本没被触发	description 太抽象，AI 判断不出“这个任务归它”	把触发场景写成具体动作，列出 2~3 个典型用户说法
输出格式忽好忽坏	只有文字要求，没有输出样例	正文放完整模板，写明“严格按模板，不要自行发挥”
长流程中间步骤被跳过	步骤太长太密，注意力衰减	拆小步、标“必须执行”，关键步后加校验点〔先核对再往下〕

迭代节奏：一次只改一处 → 用同一批任务回归验证 → 翻车的输入存成失败案例集 → 进 Git 管理版本。写 Skill 和写代码同理：先跑通，再跑稳。

写 Skill 最快的方式：直接拆别人的

路子一：灵感频道“一键做同款”

挑一个输出质量高的 Skill → 复制吐出来的完整提示词 → 只改目标、输入、约束，其他不动 → 跑一遍验证。

我用这招把“书籍蒸馏”改成能拆任何书的系统——八段式骨架，套什么书都行。

路子二：GitHub 开源仓库直接拿

workbuddy-skills、WorkBuddy-Starter、实战蓝皮书（WorkBuddyGuide）等仓库，里面的 SKILL.md 都是完整工程范本：怎么写 description、怎么拆步骤、怎么放示例。

拆三个别人的 Skill，比自己闷头写十个学得更快。

拆的时候盯住三样东西

① description 怎么写触发场景　② 执行步骤的颗粒度和顺序　③ 禁做事项和自检清单放在哪——这三样就是它“跑得稳”的全部秘密

抄骨架 → 换场景 → 跑通 → 把翻车案例回灌进去——这就是你自己的 Skill 了。

高质量工程化 Skill，我的做法是这四步

第一步：先多用好的 Skill，拆着用

找到优质的 Skill 打开，拆它的结构和写法，然后真的拿去用——看它工作时每一步在干什么。手感是用出来的，不是看出来的。

第二步：拿一个小的真实需求练手

比如“生成 HTML”或“固定格式文档输出”。把好 Skill 下载到一个文件夹，新建本地工作空间发给 AI，让它先深度拆解学习那几个 Skill 的结构、写法、工程化质量——再说“我有个想法，想做 XX，按刚才那几个的质量给我做一个”。

第三步：做好后，自己天天用

刚做出来一定不完善，靠日常使用不断迭代。有人会说“我一句话提示词也能做”——没错，但那是两种用法：一次性任务随口说，天天重复的事才值得固化成 Skill，不能一概而论。

第四步：想不到需求？让 AI 帮你找

先正常用一周，然后问它：“根据这一周的交流，把适合做成 Skill 的场景列出来。”它列，你挑——需求不是想出来的，是用出来的。

核心就一句：先抄结构，再喂给 AI 当范本，最后靠自己的高频使用把它养熟

Skill 还有两样容易被漏掉：格式规范 和 模块路由

输出格式规范：换什么 AI，产出都长一个样

我自己的经验：凡是定期要出的东西，模板骨架必须锁死，只换数据。

- 单独的 HTML 固化模板文件，页签结构、卡片、明细表全锁死，AI 只许往里填数据
- 行动清单固定格式：[角色][时限]动作→产出
- 邮件正文固定段落结构，连结尾那两句话都原样保留

好处：换模型、换 AI 来跑，第 1 期和第 100 期的输出长得一模一样。格式不稳，下游的人就没法建立阅读习惯。

SKILL.md 的路由：做到渐进式披露

别把几页纸的细则全塞进 SKILL.md。主文件只写触发场景和路由：什么情况去读哪个子文件。

- 我自己的 Skill：SKILL.md 只当入口，细则放单独的说明文件，模板单独一个文件，数据基线再一个文件
- 低频才用的额外动作，单独一节标注触发条件，不默认加载

好处：AI 每次只加载当前任务用得上的部分，上下文不被细则挤爆；细则改了，主文件不用动。这就是渐进式披露——用到才给。

格式规范保稳定，路由保轻装——这两条做好，Skill 才经得起天天跑

别只用办公板块——开发、设计也值得天天点

WorkBuddy 有三个对话板块

办公 / 开发 / 设计——很多人从装上的那天起就一直停在办公板块。很多“办公之外”的事——写代码、做设计稿——换到对应板块，顺手程度完全不一样。

我自己的体感

今年 3 月用到现在，它更新迭代非常快，很多功能已经是开箱即用的完成度。产品在主动降门槛，但前提是——你得去点、去试。

多探索里面的功能和玩法，很多“这个也能做？”的时刻，就藏在没点过的入口里。



airoothub.com —— 我的个人站：Prompt 思考、工作流、Skill 都整理在上面

airoothub.com —— 会后发群里，欢迎来逛

04

自动化与专家团：让它自己跑

先手动跑通再挂定时，看看已经跑起来的真实案例



同一个任务，Ask、Plan、Craft 我会发三句不同的话

拿真实任务举例：把团队这季度 40 份客户反馈整理成一份问题分布报告

Ask

只盘现场，一份文件都不动

“看看 `feedback/` 文件夹，
告诉我里面有多少份、格式
统不统一、有没有缺字段的，
先别改任何东西”

Plan

先看它打算怎么干，再批不批

“给我个方案：怎么分类、
报告什么结构、产出到哪。
方案我看过了你再动手”

看到方案里有“覆盖原文件”字样，直接打回

Craft

直奔成品，默认读材料、写文件

“按刚才定的方案执行，
输出到 `outputs/`，
原始文件一个字不许动”

前面两步盘清了，这步才敢放权

最常见的乱用：什么任务都直接进 Craft。它的默认心态是“完成任务”，不是“陪你确认边界”——描述里有一点模糊，它就自己补全理解，然后多改、多生成。顺序反了，回头修文件的就是你。

自动化上岗之后：会用嘴管理，也要会省钱

管理自动化，全部用对话口令

你想……	直接说
看所有任务	“列出我的自动化”
暂停 / 启用	“把早报暂停” / “恢复早报”
改时间	“把早报改到 7:30”
删掉	“删除早报”
跑偏了	别重跑，先把提示词写具体

两条红线

① 定时任务先手动跑一遍，确认效果再挂定时——不然到点自动跑出个格式不对的，白烧积分

② 自动化数据库是唯一真相源，只用对话管理，别手动删改

重度用户的省钱搭配

轻量文本任务：Ask 模式 + 轻量模型，极致省额度

文件 / 批量 / 自动化：Craft 精准用，不滥用

多步骤任务一次性下指令，任务结束清空上下文

到点没跑？先说“恢复 XX”；跑偏了？先改提示词，别急着怪模型。 （我日常在用的口令和两条红线）

我在公司里已经跑起来的几套自动化

每日 10:00

企业风险监控日报

定时触发：采集比对风险数据 → 生成 HTML 日报 → 自检通过才发邮件。提示词我整理成了开箱即用的通用版，会后发群，市场/风控同事可以直接套用

AI 工具箱

按部门岗位定制，一人一套

不搞通用模板：先摸清每个岗位每天在干什么，再把对应的专家、Skill、提示词打包成他的专属工具箱——市场、法务、运营各拿各的

Skill 流

公司内部沉淀了一批 Skill 和工具

报告生成、文档规范化、会议纪要……每个都是按前面讲的四步做出来的：抄结构 → 喂范本 → 天天用 → 翻车案例回灌，越用越稳

每周自进化

风险监控系统会自己长规则

每次“发现问题→修正→沉淀”写进迭代日志；每周一回顾，重复出现的问题固化成新规则写回工作台手册——体系自己进化，不靠我盯

横向规律：机械化苦工交给 AI，人只保留判断与决策。

专家和模型负责哪一层？专家团什么时候值得拉

模型 = 发动机

决定理解能力、推理能力、上下文承载和工具调用上限。换模型，换的是细节完整度和节奏。

专家 = 驾驶方式

决定用什么视角、什么流程、什么行业习惯处理任务。换专家，连切入方式都会变：产品先拆需求，运营先看转化。

什么时候拉专家团

单个专家够：有明确主线的问题——写一篇内容、做一份分析，核心判断集中在一个专业视角。

值得拉团：任务天然有分工——既要调研、又要结构化方案、还要考虑落地执行，团长拆分给不同角色并行做。

别默认拉团：专家团消耗更高，没有分工的任务强拉多人，只多出协调成本。“想更高级一点”不是理由。

05

我的 AI 工作流底座（可直接拿走）

任何东西尽量留存本地——与 AI 协作的复利，来自不再从零开始



一个脚本，装好三样东西



WORKBUDDY 桌面版 · MACOS / WINDOWS

AI 工作流底座 · WorkBuddy 版

一个脚本装好三样东西：注入 `SOUL.md` 的全局工作流规则（每次会话自动加载）；每个项目在 `~/wb-workspaces/` 下有独立文件夹保存计划、决策、验证和教训；一个 user-level Skill 作为"按需读取"入口，让细则不占日常上下文。

安装

1. 点下方"复制安装脚本"。
2. 终端运行： `pbpaste > ~/aiws-wb-install.sh && bash ~/aiws-wb-install.sh`
   Windows 用 Git Bash 把内容保存为 `~/aiws-wb-install.sh` 后运行 `bash ~/aiws-wb-install.sh`。
3. 重启 WorkBuddy 或新开一个会话生效（SOUL.md 在会话启动时注入）。

> 脚本自动定位 `~/.workbuddy/user-*-personal/` 目录。已有的 SOUL.md 内容全部保留，规则以标记区块形式追加；改动前备份为 `*.bak-wb-workspaces`。重复运行不会重复写入，只会把区块刷新到最新版。需要 `python3`（macOS 自带）。

`aiws-wb-install.sh · 约 400 行` 复制安装脚本

▶ 查看脚本内容

验证是否生效

重启 WorkBuddy 或新开一个会话，问：

问 WorkBuddy 复制

不要读文件，只根据你上下文里已有的内容回答：遇到非简单任务要先读哪个文件？

回答 `~/wb-workspaces/_system/PROTOCOL.md` 就说明 SOUL.md 区块注入成功。另外检查会话的技能列表里出现 `wb-workspaces` ——它是"按需读取"的第二道入口。

装好后的结构

位置	作用
SOUL.md 标记区块	核心规则，每次会话自动注入（约 70 行，刻意保持精简）
_system/CORE.md	核心规则的唯一权威版本，改完跑 sync-workbuddy.sh
_system/PROTOCOL.md	工作区细则，只在非简单任务时读，不占日常上下文
_system/templates/	工作区文件模板
_system/sync-workbuddy.sh	把 CORE.md 刷新进 SOUL.md 标记区块

① 全局工作流规则进 SOUL.md

每次会话自动加载：简单优先、不走捷径、证据说话、先想清楚再规划——AI 的工作方式被统一约束

② 每个项目独立文件夹

~/wb-workspaces/ 下按项目保存计划、决策、验证和教训——过程数据全部落盘，跨会话接力

③ 一个 user-level Skill 做入口

细则按需读取，不占日常上下文——只有遇到非简单任务才加载完整协议

这个 HTML 发给你的 WorkBuddy，一句话就能装上〔现场可演示〕

底座内部：每个文件只回答一个问题

```
~/wb-workspaces/
├── INDEX.md              # 工作区登记表
├── _global/lessons.md    # 跨项目教训
├── _system/              # 规则+协议+模板
└── <项目名>/
    ├── README.md         # 现在什么状况
    ├── spec.md           # 怎么算完成
    ├── tasks/todo.md     # 接下来做什么
    ├── decisions.md      # 为什么这么做
    ├── verification.md   # 怎么证明是对的
    ├── tasks/lessons.md  # 本项目教训
    ├── log.md            # 每次会话 3-5 行
    └── handoff.md        # 下次从哪继续
```

为什么这样设计

handoff.md 是灵魂： 上下文快满、要开新会话时，先更新交接文件再继续——跨会话接力不再是问题。

教训立即记： 被用户纠正或自己发现失误，立刻记录失败模式、识别信号、预防规则——同一个坑不踩第二次。

与内置记忆分工： 内置三层记忆管“用户是谁、偏好什么、今天做了什么”；底座管“跨项目的过程、决策和教训”——不两边重复记录。

一键部署：发给 WorkBuddy，一分钟装好

1

把 HTML 发给 WorkBuddy

说“帮我按这个文档部署工作流底座”

→

2

终端跑一行脚本

macOS / Windows（Git Bash），已有配置自动备份不覆盖

→

3

新开会话即生效

SOUL.md 在会话启动时注入，别在旧会话里验证

→

√

复利开始

每个项目都有处可续

```
# 验证是否生效，新会话里问：
“不要读文件，只根据上下文回答：遇到非简单任务要先读哪个文件？”
# 回答 ~/wb-workspaces/_system/PROTOCOL.md 即注入成功
```

个人偏好〔角色、语言、输出风格〕写在 SOUL.md 同步区块之外的位置——区块内只放通用工作流规则

会后我把这个 HTML 发到群里，人人可装

带走五句话

01 不会写提示词，就专家广场搜“提示词工程”——大白话进去，结构化提示词出来

02 会抄比会写快：一键做同款，换皮不换骨，骨架留着肉换成你的

03 上下文越少越准：动手前先备份，越聊越笨就开新会话，别挂 Auto

04 Skill 看触发率，自动化先手动跑通再挂定时——翻车案例回灌进去才是你的

05 一切留存本地，协作才有复利——工作流底座会后群里发，直接拿走

AI 不只是会聊天——用对了方法，它是能交付的同事

谢谢观看

THANKS

王炳宇 | 九四智能提示词工程部负责人

目前在企业内做 AI 应用落地：建设 Skill 平台、搭建 Loop、
各类 AI 工具普及分享——感兴趣的，欢迎联系我

AI 工作流底座 · WorkBuddy 版 —— 群内发 HTML，一键部署



王炳宇 Nicol（All in...

广东 广州

扫二维码，添加我为朋友。