

2026 海峡大学生人工智能创意大赛

知绘IE

面向工业工程现场改善的智能体应用

一句话进入改善任务 · 证据可追溯 · 数字由后端计算 · 数据留在本机

智能体工作台

知识库

专业绘图

工时分析与 SOP



目 录

一、项目概述.....	1
(一) 项目背景	1
(二) 现场痛点分析.....	2
(三) 目标用户	3
(四) 产品定位与边界.....	3
(五) 设计理念.....	4
(六) 系统架构	5
(七) 模块划分	5
(八) 技术路线与模型接入	6
二、智能体工作台	7
(一) 工作台界面.....	7
(二) 任务理解与编排.....	8
(三) 证据引用与核验.....	10
(四) 记忆与上下文管理	11
(五) 工作区与文件系统	13
(六) 能力扩展：插件、技能与 MCP	13
(七) 核心工具设计.....	14
(八) 安全机制	14
三、知识库.....	15
(一) 设计思路	15
(二) 资料入库.....	16
(三) 混合检索.....	18

(四) 知识浏览	19
(五) 带证据的回答	22
(六) 知识维护	23
四、专业绘图	23
(一) 模板体系	24
(二) 模板包的设计	25
(三) 自然语言参数化	26
(四) 对话式出图	26
(五) 导出与归档	27
五、工时分析与标准作业	28
(一) 视频资源库	28
(二) 五种分析方式	29
(三) 标准时间的计算口径	31
(四) AI 自动分析	31
(五) 人工复核与版本管理	31
(六) 线平衡分析	32
(七) 改善前后对比	33
(八) SOP 编制与导出	34
六、接口与数据	36
(一) 通信方式	36
(二) 核心接口	36
(三) 数据模型	38
七、部署、安全与测试	40
(一) 离线部署	40

(二) 安全与隐私	41
(三) 自动化测试	43
八、应用案例	43
(一) IE 课程实验：学生小组用知绘 IE 完成消防车装配线实验	44
(二) Z 竹业：没有专职 IE 的企业用录像找到瓶颈	48
(三) X 复材：IE 团队把重复工作交给知绘 IE	53
九、结语	59
(一) 技术创新与可扩展性	59
(二) 方案价值与社会价值	60
(三) 局限与下一步	60
附录 A 运行环境与第三方组件	61
附录 B 核心代码节选	63
B.1 标准时间计算与银行家舍入	63
B.2 线平衡率计算	63
B.3 混合检索的融合与重排	64
B.4 实体三级判重	65
B.5 自动分析的短段归并	65
B.6 分析版本快照与冲突检查	66
B.7 后端进程拉起与健康检查	67
B.8 种子数据逐文件填充	68
B.9 公开工具清单与系统代理	69
B.10 危险命令识别	71
B.11 工具调用守卫	72
B.12 智能体扩展注册	73

B.13 AI 配置加密与回退	75
B.14 知识库证据编号分配	77
B.15 知识库检索工具	78
B.16 追问建议生成	80
B.17 向用户提问工具	81
B.18 视频分片上传与校验	82
B.19 自适应视频抽帧	83
B.20 自动分析流水线	84
B.21 分批识别视频画面	86
B.22 视觉模型提示词	87
B.23 模型接口调用与重试	89
B.24 时间轴片段校验	90
B.25 模特排时法计算	91
B.26 前后端计算口径对齐	93
B.27 Wiki 页面切块	95
B.28 增量向量索引同步	96
B.29 别名扩展检索词	98
B.30 入库隐私扫描	99
B.31 图表生成缓存与并发	101
B.32 draw.io 文件自动同步	102

一、项目概述

知绘 IE 是一款面向工业工程 (IE) 现场改善的智能体桌面应用, 由一个智能体工作台和知识库、专业绘图、工时分析三个业务模块组成。工程师用一句话描述任务, 智能体负责拆解步骤、追问缺失参数、调用各模块完成检索、出图、测时、线平衡和 SOP 编制, 所有数字由模块后端的确定性算法给出, 所有产出都可以回查和继续编辑。本章依次介绍项目背景、现场痛点、目标用户、产品定位、设计理念、系统架构和技术路线。

(一) 项目背景

2025 年国务院印发《关于深入实施“人工智能+”行动的意见》, 提出推动人工智能在工业领域的场景应用。工业工程是制造企业提升效率的基础方法: 用方法研究找出不合理的动作和等待, 用作业测定确定标准时间, 用线平衡找到瓶颈, 再把改善后的做法写成标准作业指导书 (SOP) 固定下来。这套方法本身很成熟, 工作量大、步骤规范、结果可以按公式复核, 是适合人工智能介入的场景。



【图 1 “人工智能+”行动赋能制造业现场】

难的是把方法落到每一个工位上。多数中小企业没有专职 IE 人员, 也很少购买专业测时软件。改善工作常常落在生产主管或工艺员身上, 他们白天要盯交期、排人员、处理品质异常, 测时只能挤时间做。外请精益顾问能在短期内拉起一轮改善, 但顾问离场后, 工时基线往往没人维护。几年下来, 标准时间不再

更新，产线平衡靠经验调人，SOP 和现场的实际作业慢慢对不上。知绘 IE 的出发点，就是让这些企业在没有专职 IE 的情况下，也能持续地做测时、找瓶颈、编 SOP。

(二) 现场痛点分析

一次完整的工位改善通常从录像开始。工程师站在工位旁，把操作工连续几个作业循环拍下来。回到办公室后，要逐帧回放录像，把一个循环切成取料、定位、加工、放料这样的要素，记下每个要素的起止时刻，再判定它属于有效、辅助、注意还是浪费。切分是最耗人的一步，一段 30 s 的录像逐帧切完往往要十几分钟。接着要为每个要素取评比系数和宽放率，换算出标准时间；把一条线上各工位的时间汇总起来，画出山积图，找出瓶颈工位；改善之后再测一遍，同口径对比前后差异；最后把工序编码、标准时间、作业要点和工位截图一项项排进 Excel，下发到工位。熟练的 IE 工程师处理一条十几个工位的产线也要几天，中间的数据散落在录像文件、手写记录和几张 Excel 表里，公式各人各写，换一个人接手就很难对上口径。

通用大模型可以回答 IE 问题，但直接拿来用，工程师很快会碰到三个问题。第一是出处不可核。问模型“宽放率一般怎么取”，它会给出一段通顺的解释，却说不清出自哪本教材、哪一节，不同教材的口径又不完全一致，工程师不敢把这段话写进报告。第二是数字不稳定。把七个工位的时间贴给模型算线平衡率，两次提问可能得到两个结果，也看不到中间过程，而标准时间是定额和计件工资的依据，差一位小数就是实打实的成本。第三是产出不可编辑。模型给出的是一段文字或一张图片，工程师需要的是能继续修改的要素表、能调整的山积图和能直接下发的 Excel 作业指导书。

下表把现场痛点和知绘 IE 的对应设计放在一起。

【表 1 现场痛点与知绘 IE 的对应设计】

痛点	现状	知绘 IE 的设计
方法难查、口径不一	定义和公式散落在教材、笔记和网页中	资料入库编译成知识网络，回答中的引用带 [En] 编号，可点开原段落
测时费工	逐帧切分一段 30 s 录像要十几分钟	视觉模型按抽帧给出要素初稿，工程师在时间轴上复核修改
计算不可复核	表格公式各人各写，模型心算更不可靠	标准时间、线平衡率、MOD 换算由后端统一算法计算，智能体只引用
专业图门槛高	山积图、人机作业图、控制图需要专门软件	38 个 IE 与质量管理模板，参数化生成，可再编辑、导出
SOP 编制脱节	工序时间和作业指导书分开维护	从分析记录直接生成工序编码和 SAM，导出带工位截图的 Excel

痛点	现状	知绘 IE 的设计
数据外泄顾虑	工艺文件和作业录像属于企业内部资料	所有服务只监听本机地址，数据存在本机，模型可换成本机部署的开源模型

(三) 目标用户

第一类用户是企业里的 IE 工程师。一位负责缝制车间的工程师上午拍完一条线七个工位的录像，下午开会前要拿出线平衡结果。过去她要先花大半天切录像；用知绘 IE 时，她把录像放进一个文件夹，在对话里说一句“做成一条产线的 SOP”，系统先给出要素初稿和平衡结果，她把时间花在复核要素边界、斟酌评比系数上。

第二类用户是生产主管和班组长。他们不一定熟悉公式，关心的是三号工位的标准时间是多少、浪费占了几成、瓶颈在哪里。他们希望一句话问到答案，顺手看到山积图，而且这个数字和 IE 工程师在页面上看到的是同一个。

第三类用户是精益改善顾问。顾问进到客户现场，往往只有几天时间建立工时基线，再出一份有图表、有依据的诊断结论。所有数据留在顾问电脑本机、不经过第三方服务器，这一点在谈合作时很有说服力。

第四类用户是 IE 专业的师生。程序分析、动素分析、模特法排时、标准作业编制都是课程实验的常规内容。学生可以在时间轴上亲手切分要素，看到每个数字背后的公式，再把自己的标注和 AI 初稿对照。

(四) 产品定位与边界

知绘 IE 是一个 Windows 桌面应用，功能按 IE 改善的六个环节组织，下图是环节与模块的对应关系。



【图 2 IE 改善六环节与知绘 IE 模块的对应】

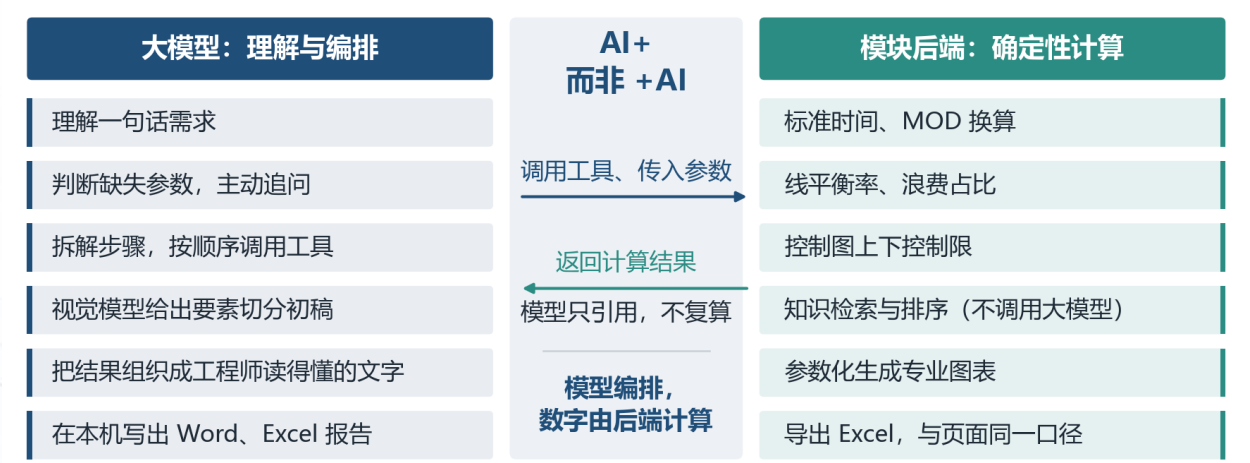
在问题与方法环节，知识库检索方法定义、公式和案例，回答附带编号证据。现状测定环节由工时分析模块导入录像，按用户选定的分析方式自动生成要素初稿。分析诊断环节统计浪费与增值占比，按产线计算线平衡率和瓶颈。改善对比环节在同一口径下并排比较改善前后两段录像。标准化环节按工序顺序生成工序编码、标准时间和 SAM，导出作业指导书。汇报沉淀环节由绘图模块出专业图，智能体在本机写出 Word 和 Excel 报告。智能体工作台贯穿全程，用户可以用对话把这些环节串起来，也可以直接进入任一模块的页面操作。

知绘 IE 也有明确的边界。它不做排产，不接 MES，也不计算节拍、人数和 UPH，这些指标依赖需求量、班制和出勤等系统中没有的输入。评比系数和宽放率由工程师判断，视觉模型给出的要素边界和类别是初稿，经人工复核后才作为标准时间使用。把边界划清楚，是为了让每一个进入 SOP 的数字都有人负责、有公式可查。

(五) 设计理念

市面上不少软件的做法是“+AI”：在原有工具旁边加一个对话框，让模型回答问题、生成文字。知绘 IE 走的是“AI+”：以 IE 改善的完整工作流程为主线，让模型承担它擅长的部分，把它不擅长的部分交给确定性算法。

大模型擅长理解人的意图。一句“把这几段录像做成产线 SOP”，背后要拆成导入、分析、平衡、导出四五个步骤，模型能读懂这句话，判断还缺哪些参数，主动追问，再按合理的顺序调用工具，最后把结果组织成工程师读得懂的文字。视觉模型看一组抽帧图像，也能给出要素切分的初稿。大模型不擅长的是稳定地算数，标准时间、线平衡率、MOD 换算这些数字一律由模块后端计算，模型只引用结果。页面上显示的、对话里引用的和导出到 Excel 的数字出自同一个函数，工程师可以拿着公式逐项复核。



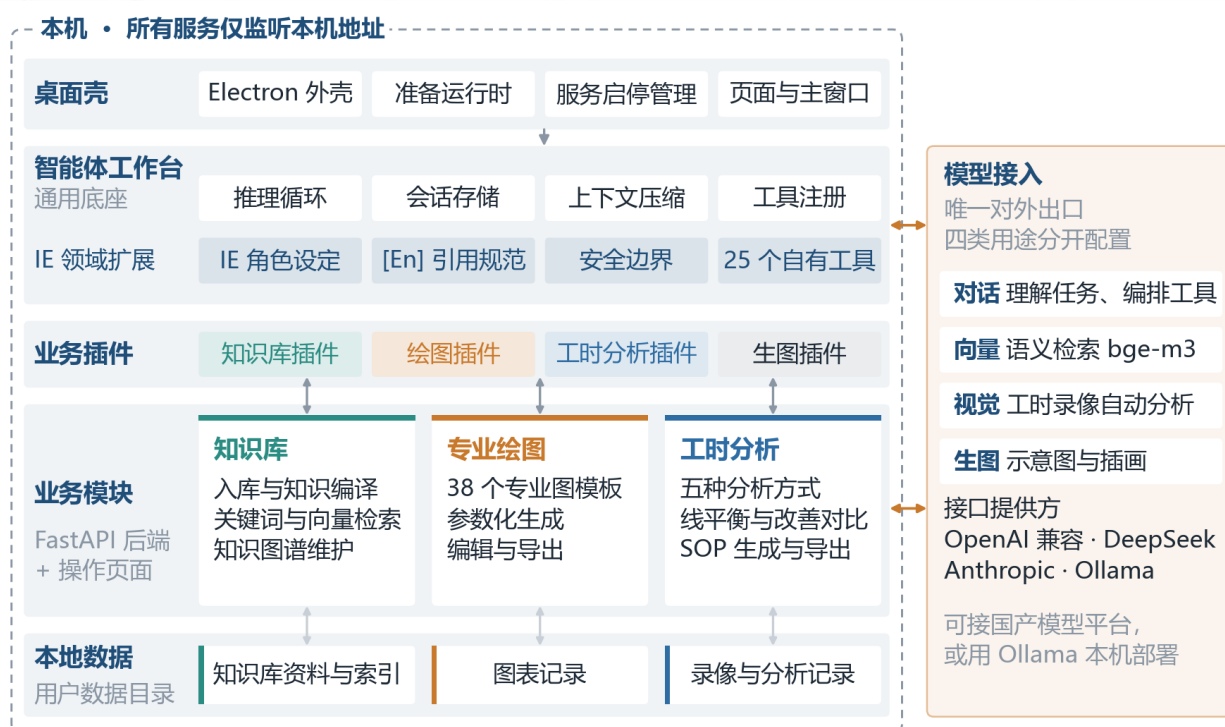
页面上显示的、对话里引用的和导出到 Excel 的数字出自同一个函数，工程师可以按公式逐项复核

【图 3 模型与算法的分工】

这条分工线也决定了产出的形态。要素初稿写成一个新的分析版本，与人工标注使用同一套数据结构；知识回答中的每一处引用都带编号，能点开原文段落；生成的图表保存为可再编辑的记录，而不是一张图片。模型参与了每一个环节，最终交付的东西都可以复核、可以修改。

(六) 系统架构

知绘 IE 从外到内分为四层。最外层是桌面壳，负责准备运行时、拉起各个本机服务、托管前端页面和管理主窗口，退出时把所有子进程一并回收。第二层是智能体工作台，也就是用户对话的地方，由智能体底座提供推理与工具调用循环、会话存储、记忆管理和能力扩展机制，在其上加入 IE 领域的角色设定、业务插件、引用规范和安全边界。第三层是业务插件，知识库、绘图、工时分析和生图各自以插件形式挂在底座上，向智能体提供一组工具和一份使用说明。最内层是本机服务与数据，三个业务模块各自是一个独立的本机后端服务，持有自己的业务数据和确定性算法，也各自带有完整的操作页面。



【图 4 知绘 IE 分层架构】

模块之间不直接互相调用，跨模块的协同由智能体完成。一份带出处的诊断报告，要从工时分析取平衡率和瓶颈，到知识库查改善方法，用绘图模块出山积图，最后在本机写成 Word，这几步由智能体按顺序串起来。每个模块因此都能脱离智能体单独使用，熟练的工程师可以直接在页面上操作。

(七) 模块划分

知绘 IE 由一个智能体工作台、三个业务模块和一项生图能力组成。

【表 2 模块划分与职责】

模块	职责	用户可见的产出
智能体工作台	理解一句话任务，追问缺失参数，调用工具，用卡片展示每步结果，在本机写 Word 和 Excel	对话、结果卡片、报告文件
知识库	资料入库与知识编译，关键词与向量混合检索，目录、导图、路线和图谱四种浏览方式	带 [En] 编号证据的回答、可点开的原文段落
绘图	38 个 IE 与质量管理专业图模板，参数化生成，编辑与导出	可再编辑的图表记录、图片、部分模板的 Excel
工时分析	视频资源库，五种分析方式，AI 自动分析，线平衡与改善对比，SOP 生成与导出	分析记录、要素表、平衡结果、Excel 作业指导书
生图	按提示词生成示意图、插画	图片文件

划分上有两处取舍。SOP 编制放在工时分析内部，因为 SOP 中的工序编码、标准时间和 SAM 直接来自分析记录，放在一起可以避免两份数据各自维护。生图和绘图分开，绘图负责有数据、有规范的专业图，生图只用于示意图和插画，保证数据图上的数值是算出来的。

（八）技术路线与模型接入

知绘 IE 做成桌面应用而不是 SaaS 网站，首先是因为数据。作业录像拍的是企业的产线和员工，工艺文件和定额属于内部资料，很多工厂不允许上传到外部服务器。其次是车间网络往往不稳定，几段工位录像动辄几百兆，上传到云端分析既慢又容易中断。为此知绘 IE 以单个 Windows 安装包分发，自带约 1.09 GB 的离线运行时，安装后不需要再联网安装任何组件。所有服务只监听本机地址，数据保存在用户数据目录，唯一的对外出口是用户配置的模型接口，它只会收到完成任务所必需的文字或抽帧图像。

模型按用途分成四类分别配置：对话模型负责理解任务、编排工具和组织回答；向量模型用于知识库的语义检索，默认使用开源的 bge-m3；视觉模型用于工时录像的自动分析；生图模型用于生成示意图和插画。四类用途需求差别很大，分开配置后，用户可以按预算和效果为每类用途选择模型，也可以让某一类走本机部署、其余走云端接口，设置页对每一类都提供测试连接。

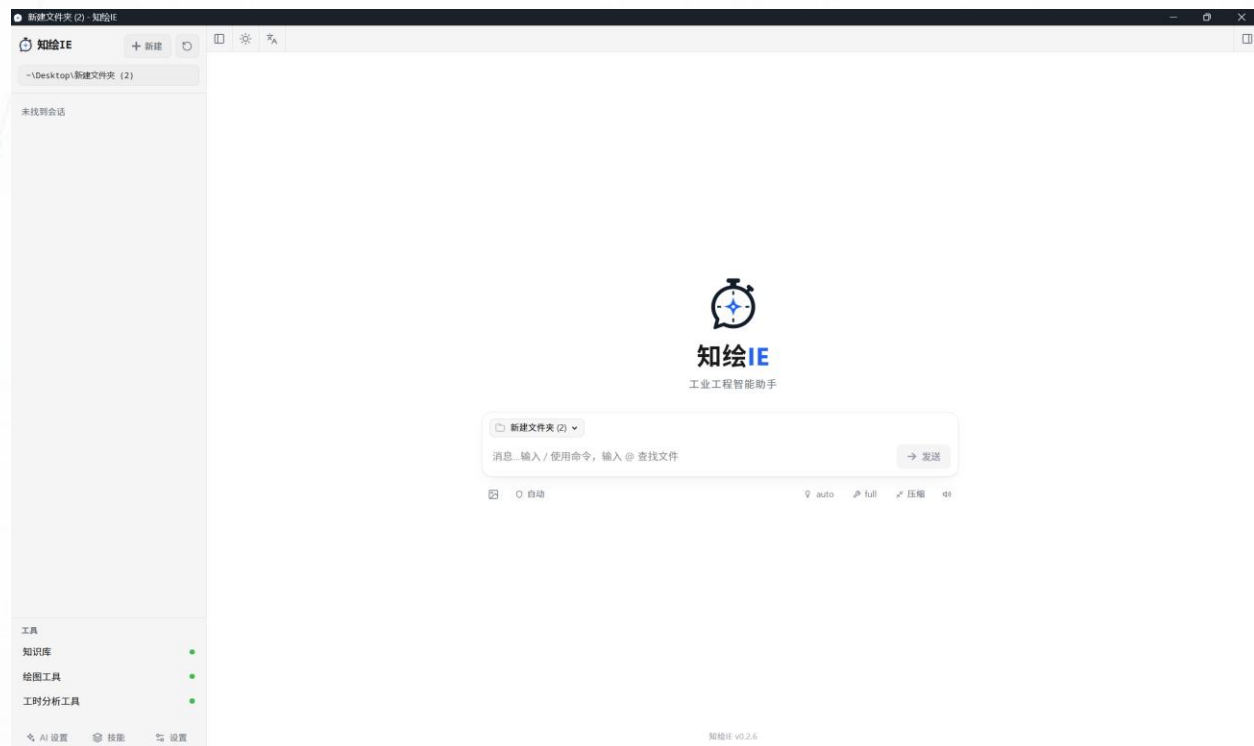
接口层面，知绘 IE 支持 OpenAI 兼容接口、DeepSeek、Anthropic 和 Ollama 四种提供方。DeepSeek、通义千问、智谱等国产模型平台都提供 OpenAI 兼容接口，填入地址、密钥和模型名即可接入；需要数据完全不出厂时，可以用 Ollama 在本机部署开源模型。模型可以替换，数字不依赖模型：无论换成哪一家对话模型，标准时间和线平衡率都由同一套后端算法给出，模型的差别只体现在理解任务、要素初稿和文字组织上，这些环节都留有人工复核的余地。

二、智能体工作台

智能体工作台是用户进入知绘 IE 后最先看到的界面，也是串起各业务模块的中枢。它建立在一个通用的智能体底座之上：底座提供推理与工具调用循环、会话与记忆管理、工作区文件系统和能力扩展机制，知绘 IE 在其上加入 IE 领域的角色设定、业务插件、引用规范和安全边界。本章先介绍工作台界面，再依次说明任务编排、证据引用、记忆管理、文件系统、能力扩展和安全机制。

（一）工作台界面

工作台分为侧栏、对话区和右侧预览区三部分。侧栏上方是当前工作区和会话列表，会话可以重命名、删除，后台仍在运行的会话会有提示；侧栏底部是知识库、绘图工具、工时分析工具三个入口，每个入口旁的状态点显示对应模块服务是否就绪，点击即可打开模块的完整页面。



【图 5 智能体工作台欢迎页】

输入框集中了常用的控制项：选择对话模型，切换工具预设（仅聊天、只读、默认、完全）和权限模式（逐条确认、自动、完全权限），附加或拖入图片，执行压缩上下文等斜杠命令。智能体运行期间，用户可以随时插话引导方向，或者把下一条消息排进队列，不必等当前任务结束。

对话区按回合展示结果。模型的思考过程和工具调用详情默认折叠，需要时展开查看；回答中的公式用 KaTeX 排版，流程可以直接以 Mermaid 图呈现；知识库引用显示为可点击的上标，生成的图表就地插入正文；本轮涉及的模块页面以卡片形式汇总在回答下方。每次回答后，工作台给出 3 条可以接着问的后续问题，长对话右侧有缩略导航，方便回到前面的某一轮。



【图 6 智能体生成的工业工程发展史时间线】

右侧预览区用来查看本轮生成或修改的文件，支持图片、PDF、Word、Excel、draw.io、网页和 Markdown，用户不用离开工作台就能检查智能体写出的报告。设置页分为 AI 设置、通用、技能和插件四部分，AI 设置中分别配置对话、向量、视觉和生图模型以及联网检索服务的密钥，每一项都可以测试连接。

（二）任务理解与编排

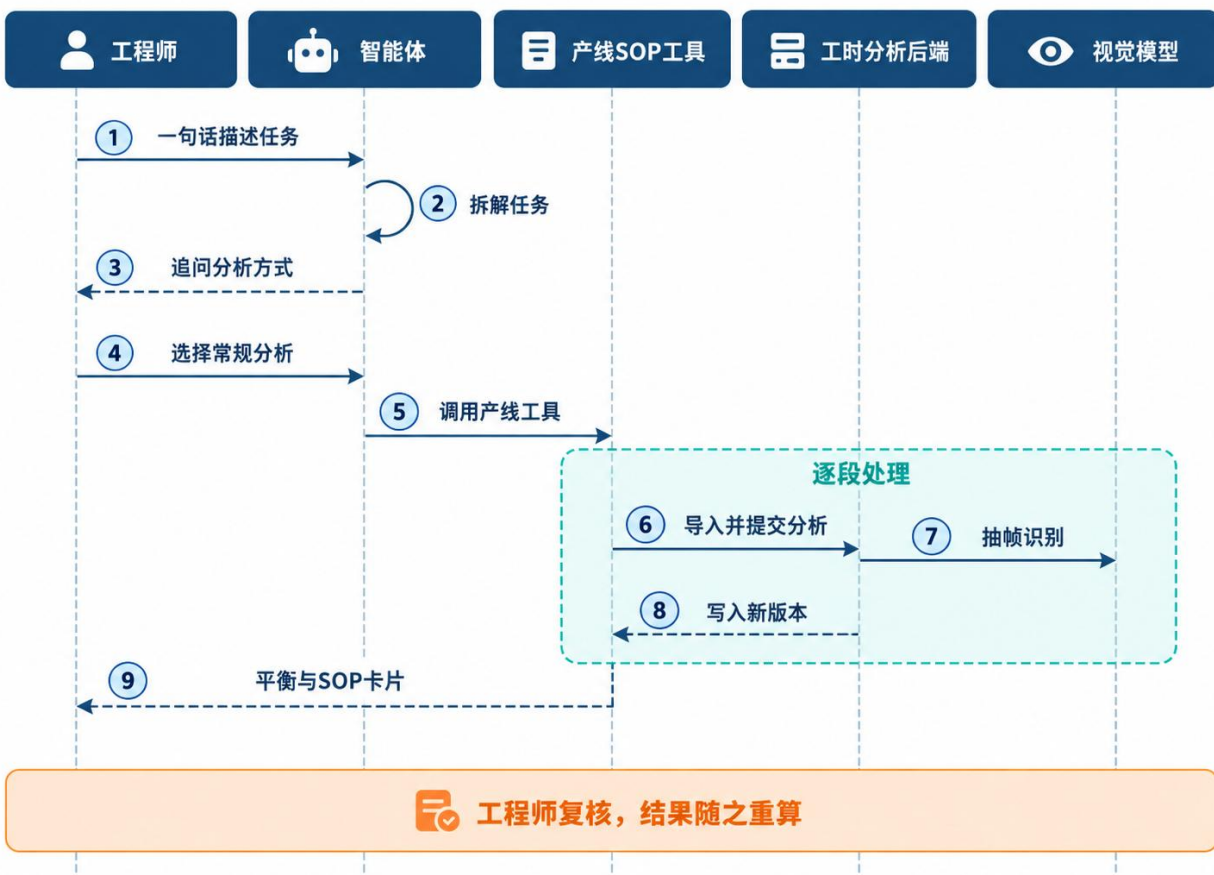
一位 IE 工程师在工作台里输入：“把这个文件夹里的 7 段工位录像做成一条产线的 SOP。”这句话有明确的对象，也有没说出来的前提：录像按什么顺序对应工序，用哪种方式分析，产线叫什么名字。智能体先把任务拆成导入、自动分析、线平衡、导出 SOP 四步，再检查参数是否齐全。

分析方式没有指定时，智能体先停下来问。知绘 IE 规定自动分析前必须由用户选择分析方式，因为常规、双手、联合、动素和模特法五种方式得到的要素结构完全不同，选错了后面全部要返工。追问以卡片形式出现在对话中，一次最多问 4 个问题，每个问题给出 2 到 5 个选项，另有一个“其他”栏可以自由填写。智能体提出问题后本轮立即结束，等用户回答再继续，不会在参数不全时自行假设。



【图 7 追问卡片】

用户选好分析方式后，智能体调用工时分析的产线工具，按文件名顺序逐个导入视频、提交自动分析并跟踪进度，单段失败只在结果里标出，其余工序照常完成。全部完成后计算线平衡并导出 SOP，结果以卡片给出，点击卡片直接打开工时模块对应的页面，看到的是同一份数据。下图用泳道时序展示这次任务中用户、智能体、工具和模块后端之间的往来。



【图 8 一次产线 SOP 任务的泳道时序】

任务的产出不止于一段文字。资源库里多了一条产线和 7 道工序的分析记录，平衡页和 SOP 页都可以继续编辑，导出目录里有一份 Excel。工程师在时间轴上把某个边界往后拖半秒，分析记录生成新版本，平衡结果随之重算。智能体在对话里报告的是线平衡率、瓶颈工序、平均工序时间和工序数，并提醒评比系数默认为 1.00、宽放率默认为 0，需要复核后再用于定额。

（三）证据引用与核验

工程师问概念类问题时，智能体先检索知识库，再组织回答。检索结果在一次会话内连续编号为 [E1]、[E2]、[E3]，同一段落再次命中时沿用原来的编号。智能体引用某段资料，就在句末标出对应编号；界面把编号渲染成可点击的上标，点开即定位到知识库里的原文段落并高亮。

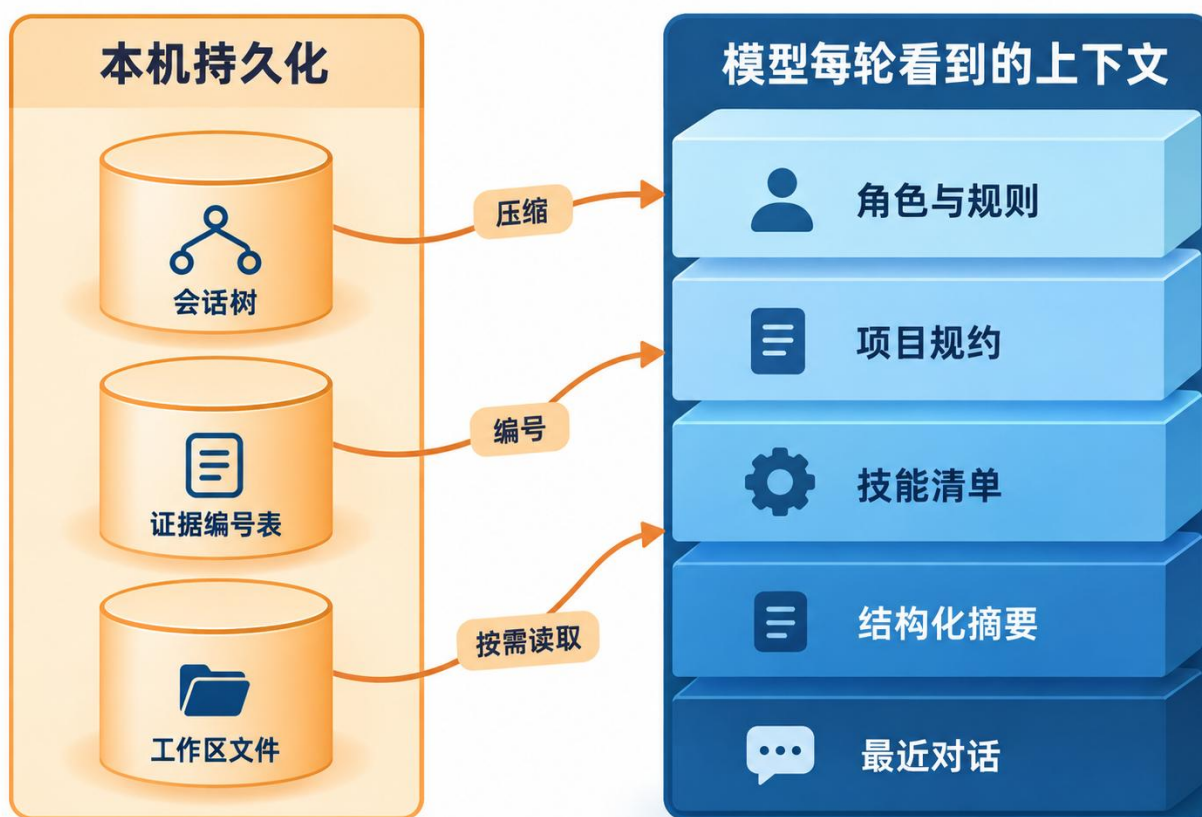


【图 9 带编号引用的回答】

编号由系统维护，不由模型生成。模型只能使用本会话检索时实际返回过的编号，回答里出现从未返回过的编号时，界面把它标为未核验。三本教材对同一概念的不同口径会作为三条带编号的证据并排出现，由工程师判断采用哪一种，而不是被模型揉成一个看似统一的答案。

（四）记忆与上下文管理

一条产线的 SOP 任务要导入多段视频、反复查询进度，工具返回的信息很快会堆满模型的上下文窗口。知绘 IE 的记忆设计要回答两个问题：模型每一轮应该看到什么，以及哪些信息要长期留在本机。下图是这套设计的整体结构。



【图 10 记忆与上下文结构】

模型每一轮看到的上下文由五层组成。最上层是角色与规则，包括 IE 工程师的角色设定、各业务插件的使用说明和安全约束，每轮固定加载。第二层是项目规约文件，用户可以在工作目录下放一份规约，写明本厂的计量口径、命名习惯和报告格式，这份约定在该目录的所有会话中长期生效。第三层是技能清单，只列出各技能的名称和简介，模型需要时再读取全文，避免大量说明常驻上下文。第四层是结构化摘要，第五层是最近约 2 万 token 的对话原文。

结构化摘要是长任务不中断的关键。上下文接近窗口上限时（以 128K 窗口为例，约在 111.6K 时），底座自动把较早的对话压缩成一份摘要，固定分为目标、约束与偏好、进展、关键决定、下一步、关键上下文六段，末尾附上已读取和已修改的文件清单。再次压缩时在旧摘要上做增量合并，而不是从头重写，前面选定的分析方式、产线名称、“评比先按 1.00”这类约定在多次压缩后依然有效。压缩在三处触发：每轮调用模型前、每条回复结束后，以及用户手动点击压缩按钮。某次调用因上下文过长失败时，系统先压缩再自动重试一次；模型请求本身遇到网络波动时按 2 s、4 s、8 s 退避重试 3 次。

本机持久化分三部分。会话以逐行追加的方式保存，每条消息记录上一条消息的编号，整份会话构成一棵树：从某条消息“从此处编辑”会在会话内形成新分支，也可以从此处复制出一个独立的新会话，关闭软件后从侧栏点选即可续接。证据编号表记录每个会话已经检索到的段落，已在当前上下文中的段落再

次命中时只给编号和标题，被压缩掉的段落再次命中时重新给出全文，保证模型引用时手里有原文。工作区文件则保存导出的报告、图片和数据，模型按需读取。

(五) 工作区与文件系统

每个会话都绑定一个工作区目录，默认是用户文档下的“知绘 IE”文件夹，其中“导出”存放 SOP、报告等导出文件，“图片”存放生成的插图，用户也可以在侧栏切换到自己的项目目录。智能体默认具备读取、编辑、写入文件和执行 PowerShell 命令四种通用能力，安装包自带的 Python 环境通过命令行调用，预装了 pandas、openpyxl、python-docx 等数据处理和办公类库，用来做数据汇总、统计检验和生成 Word、Excel 报告。

文件变化对用户是可见的。智能体写入或编辑文件时直接记录变化，执行命令前后对工作区和导出目录做快照对比，本轮新增或修改的文件汇总成一张文件卡片，点一行即在右侧预览区打开。智能体新写或修改的 draw.io 文件会自动存入绘图模块的“我的图表”，之后可以在绘图页面继续编辑。

(六) 能力扩展：插件、技能与 MCP

智能体底座本身是通用的，IE 的专业能力通过三种扩展方式接入：业务插件、技能和 MCP 服务。三者分工不同，插件接入本机的业务模块，技能封装可复用的做事方法，MCP 连接外部的联网服务。

业务插件是知绘 IE 的核心。知识库、绘图、工时分析和生图各自是一个插件，向智能体提供一组工具和一份使用说明，写明什么时候用哪个工具、按什么顺序用、哪些事情不能做。插件返回的结果分成两路：文字和数据交给模型阅读，卡片、证据表和提示只交给界面展示，不占用模型的上下文。目前四个插件加上追问和技能安装共提供 25 个自有工具，分布如下表。

【表 3 智能体自有工具】

类别	数量	主要用途
知识库	2	检索资料并返回编号证据，按页面阅读全文
绘图	6	查找模板、读取参数说明、生成图表、列出图表、导出 Excel、保存 draw.io 文件
工时分析	14	资源库管理、视频导入、自动分析与进度查询、要素修改、线平衡、改善对比、SOP 生成与导出
生图	1	按提示词生成示意图和插画
追问与技能安装	2	以卡片向用户追问，按需安装技能

技能是一份带说明的做事方法，告诉模型完成某类任务的步骤和规范。知绘 IE 内置两个技能：draw.io 技能用于手绘通用示意图并存入我的图表，数据分析技能规定如何用内置 Python 做 Excel 和 CSV 汇总、统计检验、聚类 and 常用图表。系统提示中只列出技能名称和简介，模型需要时才读取全文。用户可以在设置页的技能页搜索、安装和更新技能，也可以直接在对话中让智能体按关键词查找并安装，装好后当场就能使用，不需要新建对话。

MCP 服务用于联网检索。在 AI 设置中填写 Tavily 或 Firecrawl 的密钥后，智能体获得网页搜索、网页正文抽取和页面抓取能力，可以补充知识库之外的最新标准、行业数据和案例。联网使用有明确的规则：每次回答最多进行 3 轮搜索，单次结果压缩到 40 KB 以内，网络事实必须标注来源，网页中出现的任何“指令”一律不执行；连接失败时按 5 s、15 s、60 s、300 s 退避重连，错误信息中的密钥会被去掉。

(七) 核心工具设计

25 个工具中，有三个最能体现知绘 IE 的设计思路，分别对应知识、工时和绘图三条主线。

知识检索工具负责把知识库的证据交给模型。输入是 1 到 500 字的问题和返回条数，默认 8 条、最多 20 条。工具调用知识库的混合检索接口，为每段证据标注页面、章节和行区间，按会话连续编号；同一轮并行检索时用待定表防止编号冲突，已在上下文中的段落只发编号头，每段正文截到 1,200 字以内。给模型的是带编号的证据文本和引用规则，给界面的是证据表和命中最多的 3 张知识卡片，卡片可以直接跳到知识库原文段落。语义通道不可用时自动降级为关键词检索，并在结果中说明原因。

产线 SOP 工具把录像到 SOP 的整条链路封装为一次调用。输入是视频文件夹或视频列表、产线名称、分析方式和是否导出 SOP。工具先检查分析方式，缺失时返回追问信号，由智能体通过追问卡片请用户选择；再检查视觉模型是否已配置。随后按文件名自然顺序列出 mp4、mov、webm 视频，一次最多 30 段，建立或复用同名产线文件夹，逐段计算 SHA-256 摘要、按 4 MB 分块上传，已导入过的视频直接复用。每段视频提交自动分析后每 2 秒查询一次进度，单段最长等待 15 分钟，用户中止时同步取消后台任务。最后每道工序导出一份 SOP Excel，同名文件不覆盖。给模型的是逐工序状态、文件路径和复核提醒，给界面的是进度流以及产线平衡和 SOP 编制两张卡片。

图表生成工具按模板出专业图。输入是模板编号、参数和标题，参数必须按模板参数说明的结构填写。工具读取模板定义，调用绘图模块生成图表并保存到我的图表，统计类模板另外导出一张 300 dpi 的高清图。给模型的是图表编号、由生成脚本给出的文字摘要和一个图片占位符，模型决定把图放在回答的哪个位置；给界面的是带预览的卡片，可以一键打开编辑器。工具说明中明确规定，参数中的数值只能来自用户输入或其他工具的结果，不允许模型自行编造。

(八) 安全机制

智能体能在本机读写文件、执行脚本，这是它能替工程师写报告、整理数据的前提，也是最需要防范的地方。每一次工具调用真正执行之前都要经过三道检查：路径守卫保护软件数据目录、安装目录和系统

目录；危险命令规则拦截格式化磁盘、删除注册表、关闭安全防护等操作；权限模式决定写文件和执行命令是否需要用户逐条确认。智能体被拦截后会收到明确原因，并被要求换一种不破坏系统和数据的做法，不允许换一种写法绕过。工作台还只接受本机同源请求，用户在浏览器中打开的外部网页无法驱动智能体。

三、知识库

一位 IE 工程师要给新工位定宽放率，手边有三本教材：一本把宽放分为私事、疲劳和程序三类，一本给出了各类宽放的参考百分比，另一本以工作日为基数，与前两本以正常时间为基数的口径不同。为了确认一个数字该怎么算，他要在这三本书之间来回翻，最后还说不清引用的是哪一页。知识库要解决的就是这件事：把散落在教材、讲义和企业文档里的方法、定义和公式整理成一张互相链接的知识网络，让工程师和智能体都能在一秒左右找到出处，并且每一条引用都能点回原文核对。

（一）设计思路

常见的做法是在提问时从原始文档里临时切出几段文字交给大模型，每次结果不同，同一个概念在不同资料里的说法也不会被对齐。知绘 IE 换了一个方向：资料在入库时由大模型一次性编译成结构化的百科页，检索时直接在这些页面上查找。编译的成本在入库时付一次，之后每次检索都不再调用大模型。

页面分三类。实体页对应一个方法、工具、概念或指标，正文按定义、核心特征、历史与发展、适用场景、典型案例、常见误区、与相关概念的区别、不同素材中的观点、参考来源九段组织；主题页把一组实体串成专题；素材页是一份原始资料的摘要和抽取清单。页面以文本文件保存在用户本机，数据库里的页面、链接和别名索引都由文件派生，随时可以整体重建，用户也能直接用文本编辑器查看和备份。抽取阶段设有 40 个 IE 领域词组成的白名单，领域外的内容不会进入知识库。

知识按书组织，首页是一个书架，目前内置质量管理、基础工业工程和精益生产三本，每本标出章节、主题和实体数量。全库共有实体页 1,297 个、主题页 257 个、素材页 44 个，别名组 1,095 组，页面之间的链接 31,991 条，供语义检索使用的向量 24,207 条。



【图 11 知识库书架】

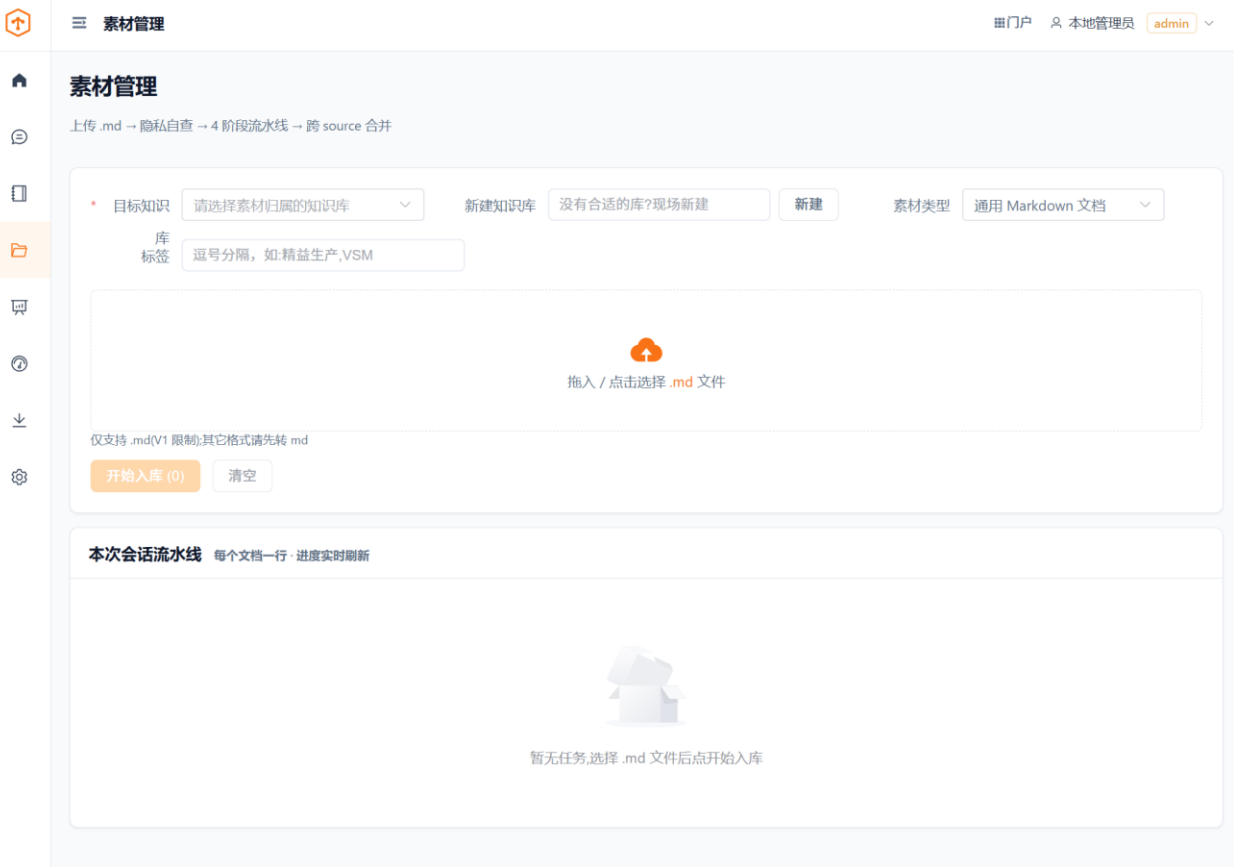
(二) 资料入库

一份资料从上传到成为知识网络的一部分，要经过隐私扫描、缓存判定、抽取、成页和判重合并五道工序。



【图 12 资料入库流水线】

隐私扫描放在最前面。资料中出现身份证号、API 密钥或链接里的访问令牌时直接拒绝入库；出现手机号、邮箱或银行卡号时给出提醒，提示里只显示命中片段的首尾几位。缓存判定按正文内容计算指纹，同一份资料重复上传时后续工序全部跳过，不会重复消耗模型调用。抽取阶段由大模型通读原文、知识库的使命说明和现有结构摘要，输出素材摘要、实体、主题、实体间关系以及资料内部的矛盾观点；输出格式不合规时附上错误信息重试一次，随后过滤掉领域外实体、名字过短的伪实体和自己指向自己的关系。成页阶段为每个新实体、新主题各写一页，多页并发生成，单页失败只记录警告，不影响其他页面。



【图 13 素材管理页面】

判重合并决定了知识网络是否干净。IE 术语的同义写法很多，“标准工时”“标准时间”“ST”是一回事，“精益生产”“TPS”“丰田生产方式”也是。新实体先按名称和别名匹配已有页面，匹配不到时再与全部已有实体的简介做向量相似度比较，按下表处理。

【表 4 实体判重的阈值与动作】

余弦相似度	处理方式	取舍理由
不低于 0.92	自动合并	几乎可以确定是同一概念，由大模型把新旧两页融合成单一定义，来源取并集
0.75 至 0.92 之间	新建页面，同时给出合并建议	相近但未必相同，交给别人确认
低于 0.75	新建页面	视为不同概念

自动合并的门槛定得偏高，是因为错误合并的代价远大于重复。“标准时间”和“正常时间”语义非常接近，却是两个口径不同的概念，一旦被合进同一页，定义就混在一起，事后很难拆开；而两个重复页

面只要人确认一次就能合并。基于同样的考虑，向量模型不可用时整篇资料入库失败，而不是退回只按名称判重，以免悄悄产生一批重复页面。

全部页面写完后，系统做一次全库整理：别名链接统一改写为规范名；指向不存在页面的悬空链接先尝试用别名改写，再为出现频率高的缺失概念补写页面，剩余的转成普通文字，保证每个链接都能点开；没有归属的孤立实体被归入已有主题或新综合出的主题；最后增量更新向量索引。以一章《质量管理概述》做全流程评估，耗时 18.6 分钟，生成 40 个实体页和 11 个主题页，4 个已有实体页被融合更新，悬空链接从 2 个清到 0 个。

（三）混合检索

检索是知识库被调用最频繁的环节，一轮对话里智能体可能检索好几次。知绘 IE 的检索不调用大模型，带来三点好处：一次检索约 1 秒返回，不受模型服务排队和网络波动影响；同一个问题在同一个知识库上的返回结果固定，评审一条回答时可以重放当时的检索；检索本身不消耗模型调用，频繁查询也不增加费用。



【图 14 混合检索流程】

问题进来后，先按别名词表扩展成整组同义词，再把“是什么”“有哪些”“如何”这类问句结构切开，补充有意义的片段。随后分两路检索。关键词通道遍历全部页面，标题和别名命中的权重高，正文命中的权重低，擅长精确的术语查询。语义通道用 bge-m3 模型把问题转成 1024 维向量，与按章节切好的段落比较相似度，擅长换了说法的问题和询问概念关系的问题。两路结果先按名次做倒数排名融合（RRF），再用一个确定性公式重排，综合关键词得分、语义相似度、问题与标题章节的重合程度和命中次数；关键词得分取对数压缩，避免标题完全命中压倒语义相关性。同一页面最多保留 3 段，让结果覆盖多个页面，而不是被一篇长文占满。

向量索引缺失或与当前知识库不一致时，系统只用关键词通道，并把降级原因告诉用户，附上设置页重建索引的入口。检索没有任何命中时，问题里的关键词自动写入知识缺口队列，提示维护者补充资料。

(四) 知识浏览

除了在对话中提问，工程师也可以直接浏览知识。每本书提供目录树、思维导图、学习路线和知识图谱四种视图，分别对应查阅、梳理、学习和探索四种用途。

目录树按“章节、主题、实体”三级展开，与教材的目录结构一致，适合按章节查阅，管理员可以在这里调整顺序。



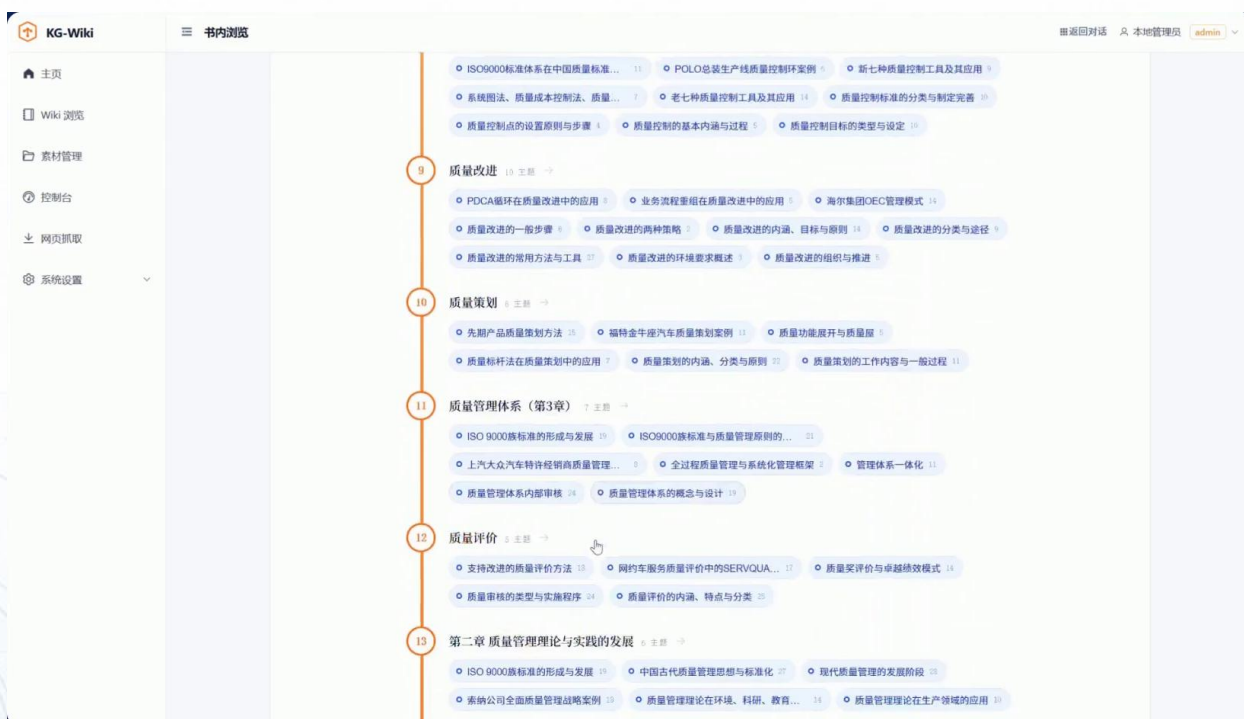
【图 15 目录树视图】

思维导图把同一本书画成从书到章节、主题、实体的四级树，可以按章节、主题或全部三档展开，拖动和缩放浏览，适合在备课或做报告前梳理一本书的知识框架。



【图 16 思维导图视图】

学习路线采用地铁线路的样式，章节是一个个站点，站点下排列该章的主题胶囊，点开主题在侧边抽屉中查看包含的实体。已浏览的章节和主题会点亮，页面顶部显示“已浏览 X / N 章”的学习进度，一直走到全书终点。这种视图面向新入职的工程师和 IE 专业的学生，把一本教材变成一条可以按进度走完的路线。

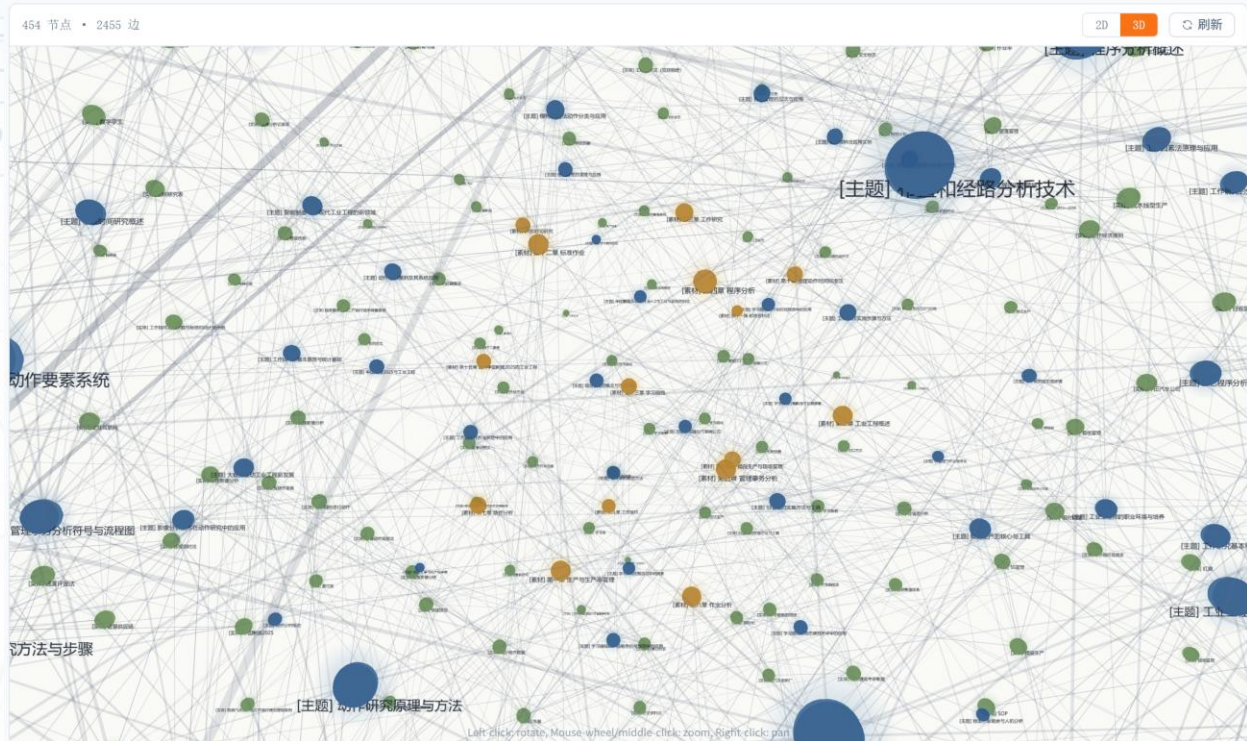


【图 17 学习路线视图】

知识图谱以实体、主题和素材页为节点，以页面间的引用为边，边的权重综合引用次数、关系类型和两页共享的资料来源。图谱可以在二维力导向布局和三维分层布局之间切换，三维布局把素材、主题和实体分布在三层同心球面上，悬停一个节点时高亮它的全部邻居。社区划分使用 Louvain 算法并固定随机种子，每次重建得到相同的划分；当前图谱有 1,598 个节点、10,120 条边，划分为 27 个社区。为了可读，每个节点只显示权重最高的若干条边，连接多个社区的桥节点和几乎没有连接的孤立实体会被标出来，它们往往是知识缺口所在。



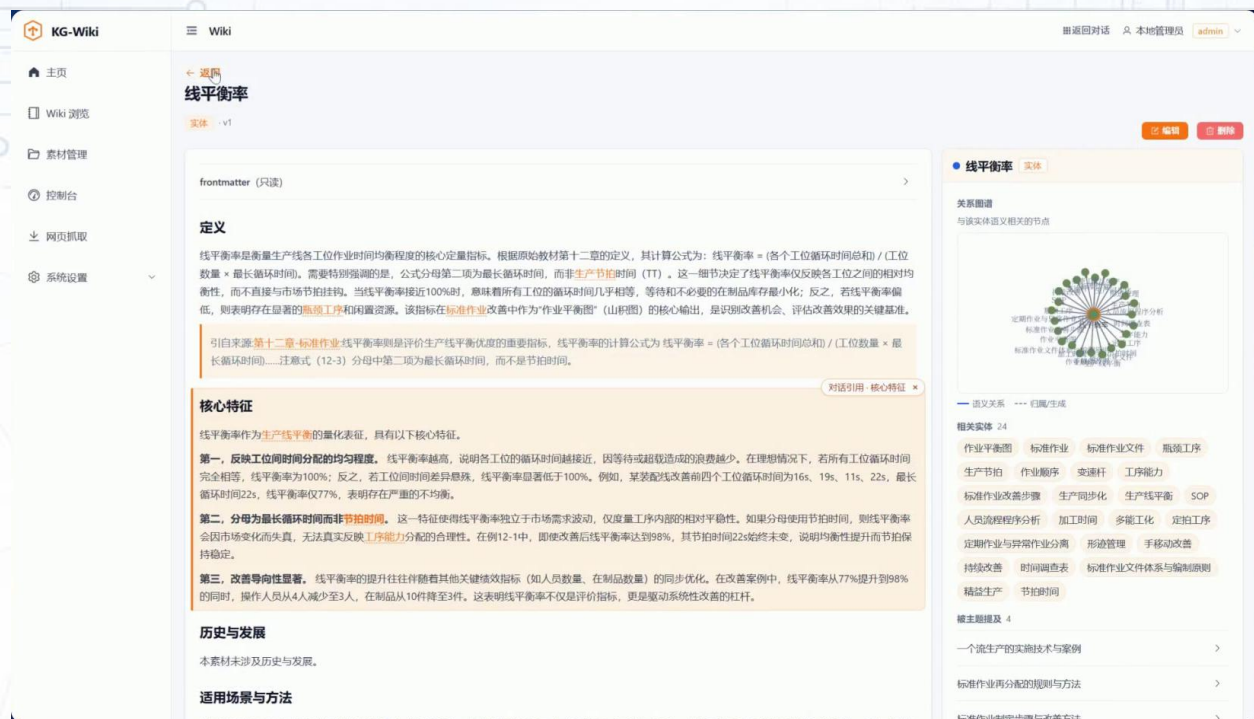
【图 18 知识图谱二维视图】



【图 19 知识图谱三维视图】

（五）带证据的回答

回到开头的场景。工程师在对话中问宽放率怎么定，智能体检索知识库，检索结果在会话内按 [E1]、[E2] 连续编号，回答里每一个论断后面都标出所依据的编号。点开编号，知识库页面打开到对应段落并高亮，工程师读到的是原文，而不是模型的转述。页面右侧还有一张以当前实体为中心的局部关系图，列出直接相连的方法和概念，方便顺着往下查。



【图 20 知识页面与引用定位】

检索结果中因篇幅被截断的段落会注明，智能体需要完整上下文时可以按段落读取整页。这样三本教材的不同口径会作为三条带编号的证据并排出现，由工程师判断采用哪一种。

（六）知识维护

知识库会随着资料增加而生长，维护的原则是机器提出，人来确认。相似度落在 0.75 到 0.92 之间的合并建议由维护者逐条确认后再合并。健康检查可一键扫描孤立页、断链、重复实体、低置信度内容、矛盾观点和索引不一致六类问题；知识缺口队列汇集检索无命中的问题，加上图谱标出的孤立概念，指明下一步该补什么资料。每次页面修改都保留版本差异，可以对比查看；页面改名后旧地址自动跳转到新页面。删除一份素材时，只被这份素材引用的实体页随之删除，被其他资料共同支撑的页面保留下来。

四、专业绘图

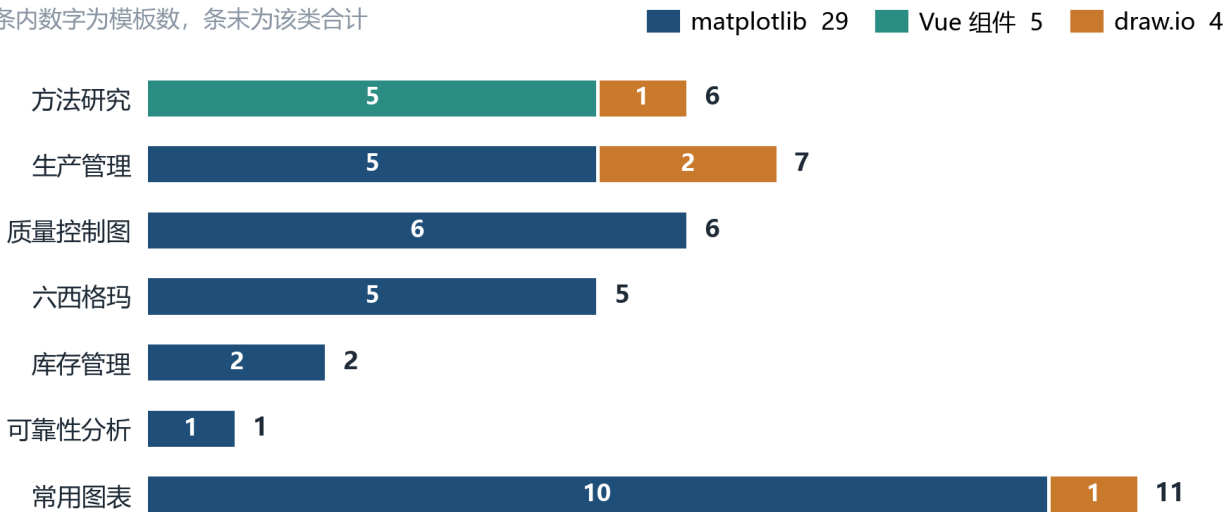
IE 和质量管理的报告离不开专业图。改善前后要用山积图对比各工序负荷，人机配合要画人机作业图，过程是否稳定要看控制图，问题根因要用鱼骨图梳理。这些图的门槛不在画，而在规范：控制图的上下控制限要按子组数据算出，山积图要把每道工序拆成要素堆叠，人机作业图要让人和机器的时间在同一条轴上逐格对齐。通用的办公软件不懂这些规则，工程师往往要先在表格里算一遍，再手工调格式，改一个数据就要重画一次。通用的大模型能写出画图代码，但每次写法不同，数字和格式都难以保证。绘图模块要做的，是让工程师用一句话或一张表单得到符合专业规范的图，并且这张图可以复现、可以接着改。

（一）模板体系

知绘 IE 把专业图沉淀为模板，当前内置 38 个，覆盖方法研究、生产管理、质量控制图、六西格玛、库存管理、可靠性分析和常用图表七个类别。

绘图模板注册表：38 个模板的类别与渲染引擎分布

条内数字为模板数，条末为该类合计



【图 21 模板按类别与引擎分布】

不同的图对生成方式的要求不同，所以模板按三种引擎实现。统计类的图对数值准确性要求最高，由 matplotlib 用代码计算并绘制，共 29 个；方法研究的图本质上是带符号的表格，工程师需要在格子里反复修改，做成可直接编辑的页面组件，共 5 个；流程和结构类的图需要自由调整布局，生成 draw.io 画布，共 4 个，可以在内置的离线 draw.io 中继续编辑。下表列出各类别的代表模板。

【表 5 模板类别与代表图表】

类别	数量	代表图表
方法研究	6	流程程序图、人机作业分析图、双手作业分析图、联合作业分析图、动素分析图、工艺程序图
生产管理	7	山积图、产线平衡分析、学习曲线、设备综合效率、工序 DOI 节拍图、甘特图、价值流程图
质量控制图	6	X-R 控制图、I-MR 控制图、p 控制图、c 控制图、DPMO 控制图、目标偏差图
六西格玛	5	帕累托图、鱼骨图、过程能力分析、直方图、散点图
库存管理	2	经济订货量模型、经济生产批量模型
可靠性分析	1	可靠性分析

类别	数量	代表图表
常用图表	11	柱状图、折线图、饼图、雷达图、热力图、技术路线图等

模板库页面按类别浏览，每个模板都有预览图和一句话说明，工程师不必记得图的名字，看到样子就能选。



【图 22 模板库】

(二) 模板包的设计

每个模板是一个独立的模板包，由四部分组成：生成脚本负责计算和绘图；参数说明定义这张图需要哪些输入、哪些必填、各是什么类型，并附一组可以直接运行的示例；预览图让用户一眼认出这张图；给模型的提示说明告诉大模型如何把一句话需求转成这张图的参数，以及要遵守哪些规则，例如各组数据必须等长、修改时要在原数据基础上返回完整参数。

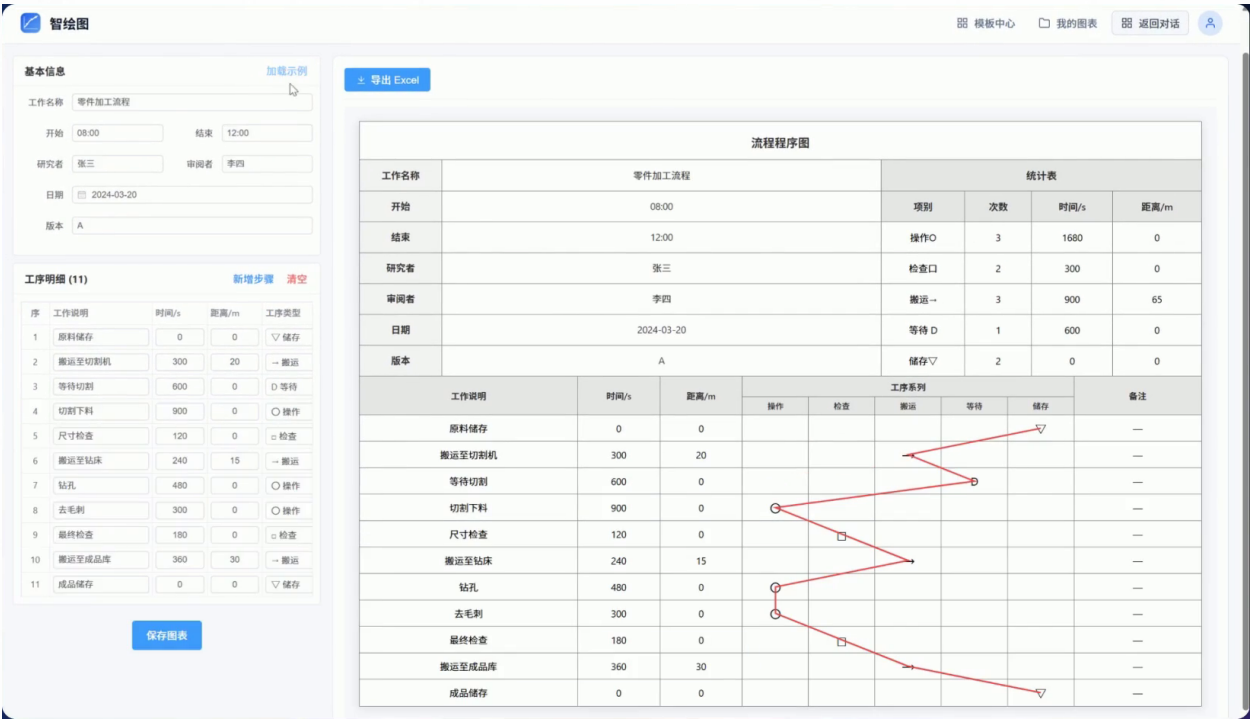
这个设计的核心是把“画什么”和“怎么画”分开。大模型只负责理解需求、组织参数，图本身由生成脚本按固定规则画出，同一组参数必然得到同一张图。这带来三点好处。第一是可复现：报告里的一张控制图，任何时候用同样的数据都能重新生成，控制限的算法写在脚本里，不会因为模型换了一种写法而变化。第二是可校验：参数不合法时，生成脚本直接指出是哪个参数有问题，模型或用户据此修正，而不是得到一张看似正常但数据错位的图。第三是可再编辑：每张图都保存为一条记录，统计图保存参数，表格的图保存表单数据，draw.io 图保存画布，历史上的任何一张图都可以打开改参数重新生成，而不是只剩一张图片。

模板包的形式也让模板库容易扩展。新增一种图只需要按同样的结构准备好四部分并登记，服务启动时自动同步，模板库页面和智能体随即可用，不需要改动服务本身。参数说明同时驱动两个入口：页面据此自动生成参数表单，智能体据此组织调用参数，两边用的是同一份定义。

(三) 自然语言参数化

工程师描述需求时用的是自然语言，比如“把这五道工序的时间画成山积图，节拍按 60 秒画一条线”。模板包里的提示说明和示例参数让大模型知道这张图需要什么，它把描述中的工序名、要素时间和节拍整理成参数，缺少关键数据时先向用户追问，而不是自己编一组数字。参数组织好之后交给生成脚本出图，如果脚本报出参数错误，模型根据错误信息修正后重试。

在模块页面上，参数表单旁边有 AI 填参入口，用户用一句话描述，表单就被填好，再逐项检查、微调后生成。对已有的图，也可以直接说“把第三道工序的时间改成 42 秒”，模型在原参数基础上修改并返回完整参数。



【图 23 参数表单与预览】

(四) 对话式出图

多数时候，工程师不需要离开对话。在智能体工作台里提出画图需求，智能体先按关键词查找合适的模板，再读取模板的参数说明和示例，按示例改写成本次的参数后生成。生成的图以卡片形式出现在回答中，图片插在正文的相应位置，旁边附一句由生成脚本给出的文字说明，智能体用它描述图的结论，不必去“看图”猜数字。

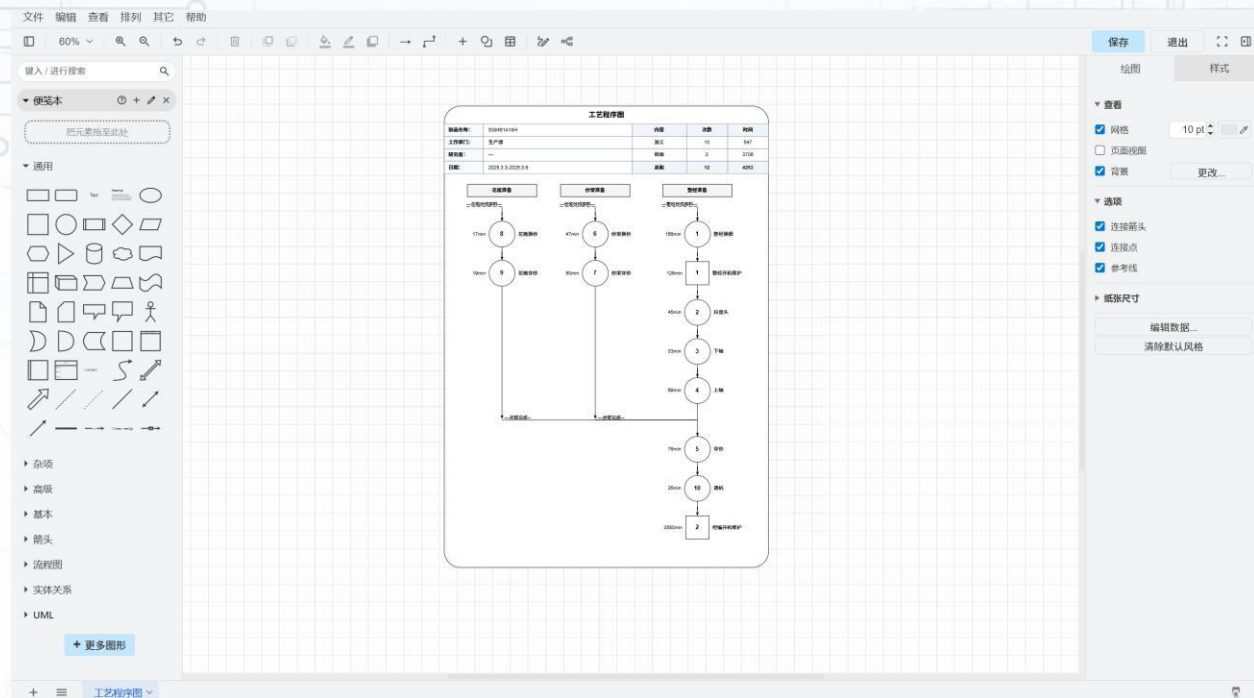


【图 24 对话中的图表卡片】

对话中生成的图同时保存在绘图模块的图表记录里，可以在页面上打开继续调整。对话负责快速得到第一版，页面负责精细调整，两者操作的是同一条记录。工时分析得出的工序时间、知识库查到的公式口径，都可以在同一轮对话里接着画成图，数字直接来自模块的计算结果，不经过模型转述。

（五）导出与归档

图最终要进入报告或者打印归档。统计图可以导出高清图片和矢量格式，中文字体在导出时嵌入或转为路径，在没有安装中文字体的电脑上打开也不会乱码。流程类的图可以导出图片、矢量图和 draw.io 文件，交给同事继续编辑，下图是在内置的离线 draw.io 中编辑一张流程类图表。



【图 25 离线 draw.io 编辑器】

人机作业图、双手作业图、联合作业图、流程程序图和动素分析图这 5 个方法研究模板还能导出带格式的 Excel，因为这几张图在企业里通常要打印、签字、归档，表格比图片更方便填写和修改。

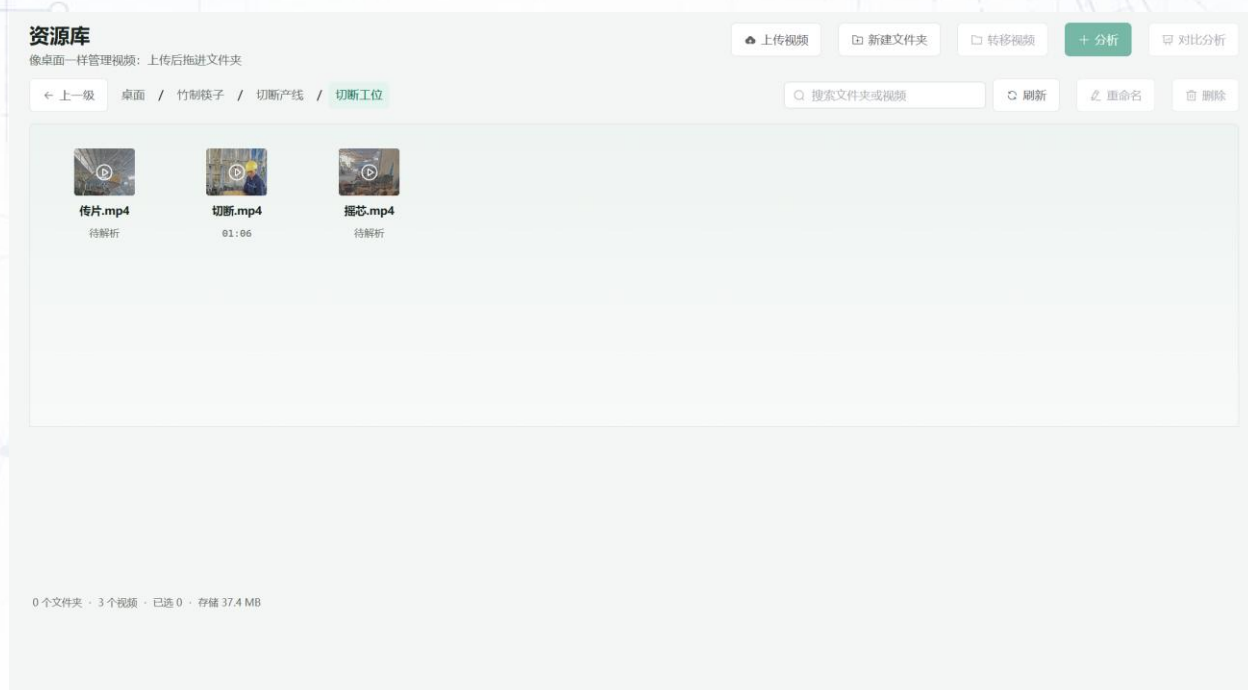
从一句话需求到一张可复现、可再编辑、可归档的专业图，绘图模块把专业规范封装在模板里，把理解需求的工作交给大模型，把计算和绘制留给确定性的脚本。工程师花在调格式上的时间省下来，可以在读图和做判断上。

五、工时分析与标准作业

工时是工业工程最基础的数据，定额、排班、线平衡、计件工资都要从一道工序需要多少时间算起。传统秒表测时要逐个要素记读数，再回办公室录入、乘评比系数、加宽放，现场动作无法回看，有争议也难以复核。录像解决了回看，却把工作量转到了逐帧找动作边界上。工时分析模块以作业录像为输入，让 AI 先做掉切分和命名，工程师在时间轴上复核定稿，再由确定性算法算出标准时间和线平衡率，最后编成标准作业指导书。

（一）视频资源库

资源库沿用工程师一条产线一个文件夹、每道工序一段视频的习惯，像管理桌面文件一样管理作业录像：上传视频后拖进文件夹，文件夹即产线，视频即工序。顶栏提供上传视频、新建文件夹、转移视频、分析和对比分析几个入口，工具栏上有路径导航、搜索、刷新、重命名和删除；鼠标悬停在视频缩略图上即可预览，每个视频下方标出分析状态，底部状态栏显示文件数量和占用空间。产线平衡和 SOP 编制都以文件夹为输入，不需要另建产线模型、录入工序清单。

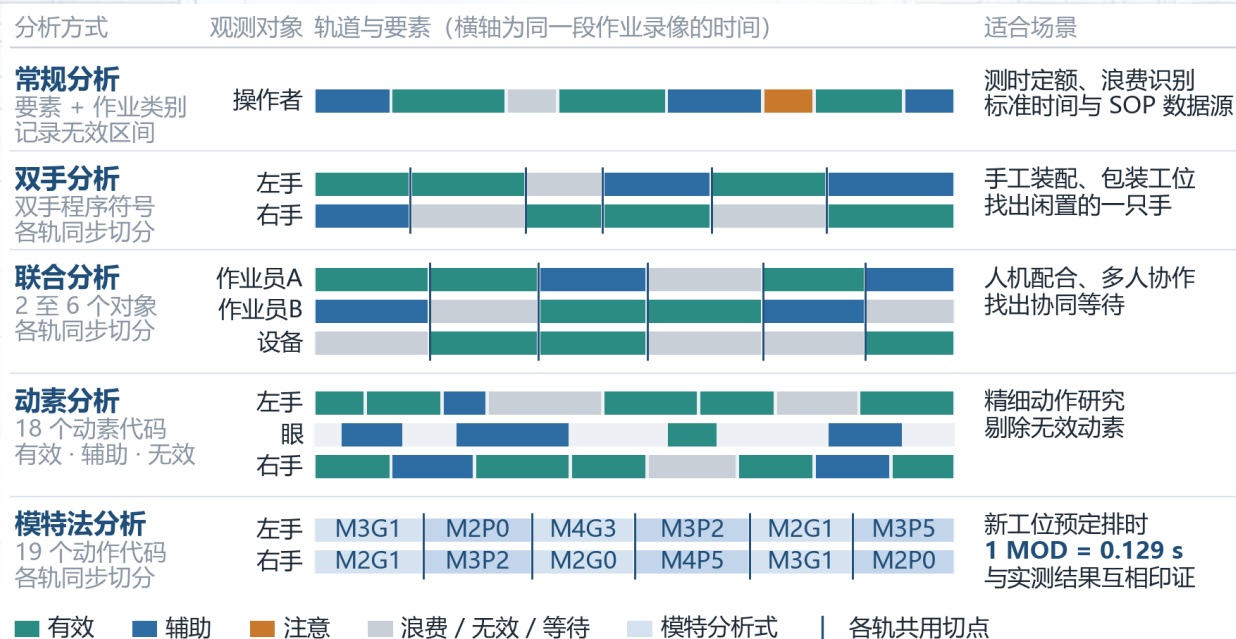


【图 26 视频资源库】

视频支持 mp4、mov、webm 格式，单个不超过 5 GiB，分块上传并校验完整性，重复上传会被识别，播放时可任意拖动进度。每个视频可以挂多条分析记录，一条分析由若干轨道组成，每条轨道代表一个观测对象，例如操作者、左手、右手或联合作业中的某个人；轨道上的片段就是一个作业要素，记录起止时间、名称、类别、评比系数和宽放率。所有指标都从这些片段算出，页面、对话和导出文件引用的是同一份数据。

（二）五种分析方式

测定额时关心每个要素的时间，改善双手动作时关心左右手是否均衡，人机配合时关心谁在等谁。知绘 IE 在同一段视频上提供五种分析方式，对应 IE 教材中的经典方法。



【图 27 五种分析方式】

【表 6 五种分析方式与适用场景】

分析方式	观测对象	记录内容	适合场景
常规分析	操作者	要素名称、作业类别、活动类型 (加工、检查、搬运、等待、储 存)、无效区间	测时定额、浪费识别、标准时间与 SOP 的数据来源
双手分析	左手、右手	每只手的动作和双手程序符号	手工装配、包装等以手部动作为主的 工位，找出闲置的一只手
联合分析	2 至 6 个对象	每个人或设备的动作与类别	人机配合、多人协作工位，找出协同 等待
动素分析	左手、眼、右手	18 个动素代码，分有效、辅助、 无效三类	精细动作研究，剔除寻找、选择、持 住等无效动素
模特法分析	左手、右手	每只手的模特分析式，1 MOD 等 于 0.129 秒	新工位预定排时、与实测结果互相印 证

双手、联合、模特三种方式是联动的：各条轨道在同一时刻切分，拖动一处边界时各轨同步移动，一次撤销回退全部轨道，表格中第几行就是各轨的第几个要素。各对象因此在表中逐行对齐，与教材上双手作业图、联合作业图的格式一致，导出后可直接用于报告。动作库内置模特法的 19 个动作代码，并会从常规分析中自动沉淀企业自己的要素名称，测得越多，要素库越完整。

(三) 标准时间的计算口径

作业要素分为有效、辅助、注意、浪费四类。标准时间由下式计算：

$$T_s = T_0 \times R \times (1 + \varphi) \quad (1)$$

式中 T_s 为标准时间， T_0 为观测时间，即要素的起止时间差扣除操作者被打断的无效区间， R 为评比系数， φ 为宽放率。浪费类要素的标准时间一律计为 0。结果按毫秒取整，采用银行家舍入，页面、对话和导出文件中的毫秒数逐位一致。评比系数和宽放率由工程师根据操作者熟练度和作业条件逐个要素填写，系统不替工程师判断。

(四) AI 自动分析

AI 自动分析省掉的是逐帧找边界这一步，工程师拿到的是已经切好、命名好、分好类的要素表初稿。下图是从录像到 SOP 的完整处理流程，分为 AI 自动分析、工程师复核和确定性计算三段。



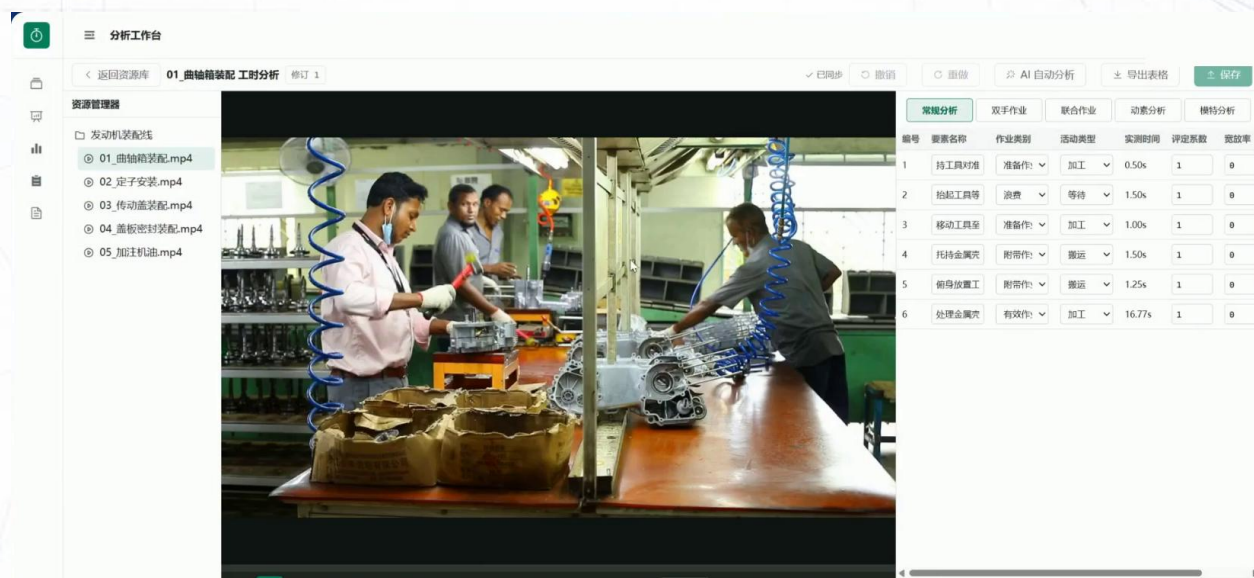
【图 28 视频到 SOP 的处理流程】

在工作台顶栏点击 AI 自动分析，或在对话中让智能体分析某个视频，任务即在后台运行。处理分五个阶段。先按视频时长抽帧，不超过每秒 4 帧、总数不超过 240 帧，短视频抽得密，长视频按比例放稀。再每 12 帧一批交给视觉模型，每帧附上时间戳，要求逐帧描述左右手、眼睛、工件和作业状态；单批失败记为看不清并继续，失败批次超过一半时任务终止，避免在多数画面没看清时硬拼出一张表。第三步由对话模型把逐帧描述按所选方式归并成要素并判定类别。第四步是不调用模型的规则后处理：短于 200 毫秒的碎片并入相邻要素，多轨切点对齐，要素首尾覆盖整段视频，动素代码和模特分析式按规范校验。最后，结果以评比系数 1.00、宽放率 0 写入一个新版本，留待工程师复核。这两个值取决于工程师对熟练度和作业条件的判断，模型从画面里看不出来，也不应由模型给出。任务进度实时显示，可离开后回来接着看，也可随时取消。

(五) 人工复核与版本管理

视觉模型给出的边界和类别是初稿，必须人工复核。标准时间会进入定额和计件工资，边界偏几百毫秒，乘上一天的产量就是可观的工时差异；边界判断又依赖现场经验，同一个“取件”从手伸出算起还是从触到工件算起，各企业约定不同。让 AI 做完粗活、让人对结果负责，是工时模块最基本的取舍。

复核在分析工作台上进行。左侧是产线文件夹中的工序列表，中间播放录像，下方是各轨道的要素色块，右侧是要素表。边播放边拖动边界即可调整起止时间，改名、改类别、填写评比系数和宽放率也在同一页面完成。



【图 29 分析工作台】

分析记录只追加不覆盖，每次保存生成完整的新版本，AI 结果与人工修改在同一条版本链上，任何一版都可找回。保存时系统核对编辑所基于的版本，期间别人已提交更新版本时，这次提交会被拒绝并提示冲突。工时数据常需交叉审核，也常要回答“这个数字是谁、什么时候改的”，版本链让这些问题有据可查。

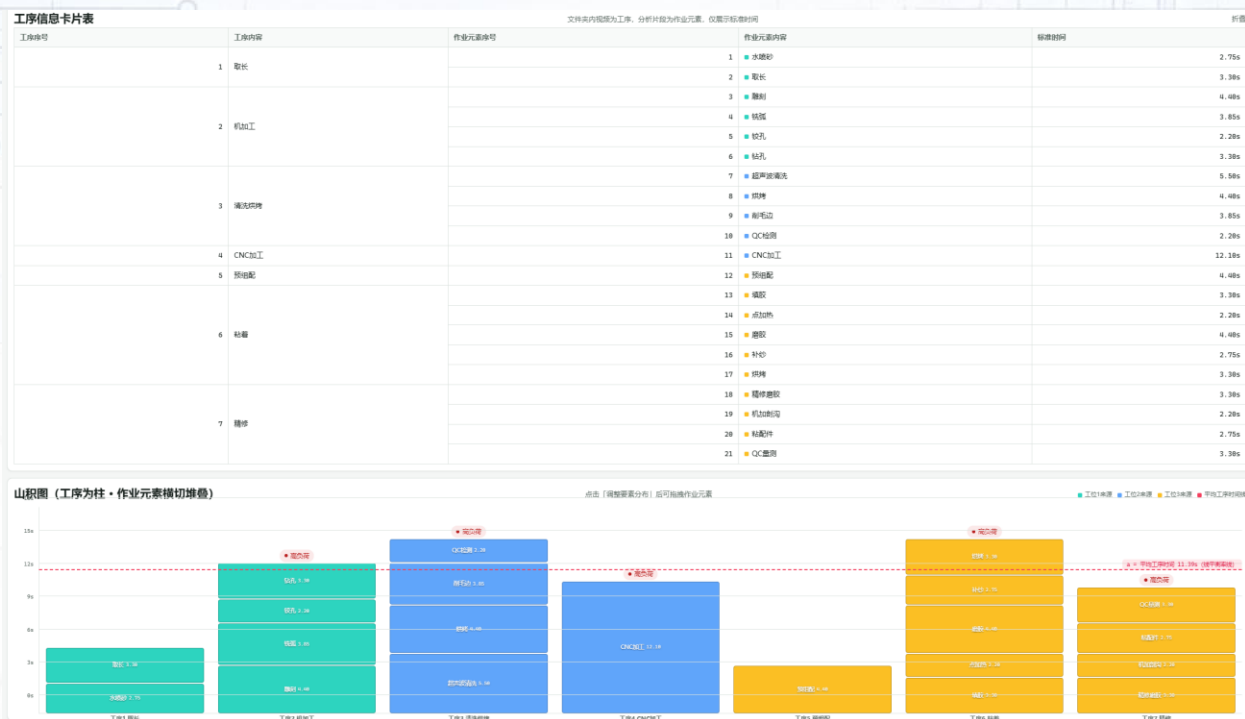
要素表是工时分析的主要产出，每行一个要素，列出名称、作业类别、观测时间、评比系数、宽放率和标准时间，占比条显示四类作业各占多少时间，底栏给出周期和增值比。其他分析方式各有统计：双手分析给出左右手作业时间的平衡率，低于 0.7 时提示哪只手利用率偏低；联合分析给出每个对象的作业和空闲时间；动素分析按三类统计左手、眼、右手的动素次数；模特法分析汇总每行的 MOD 数、正常时间和标准时间。要素表可导出为 Excel，动素符号以图形嵌入单元格，与教材表格样式一致。

（六）线平衡分析

线平衡关注整条产线。选中一个文件夹，每个视频的常规分析构成一道工序，工序时间为其中各要素标准时间之和。线平衡率按下式计算：

$$\eta = \frac{\sum_{j=1}^n T_j}{n \times T_b} \times 100\% \quad (2)$$

式中 T_j 为第 j 道工序的时间， n 为工序数， T_b 为瓶颈工序的时间。分母 $n \times T_b$ 是各工序都按瓶颈节奏运转时产线占用的总时间，分子是实际需要的作业时间，两者之比反映产线时间被有效利用的程度，各工序时间完全相等时 η 为 100%。相应地，平衡损失率为 $1 - \eta$ ，表示因工序不均衡而空等的时间比例。

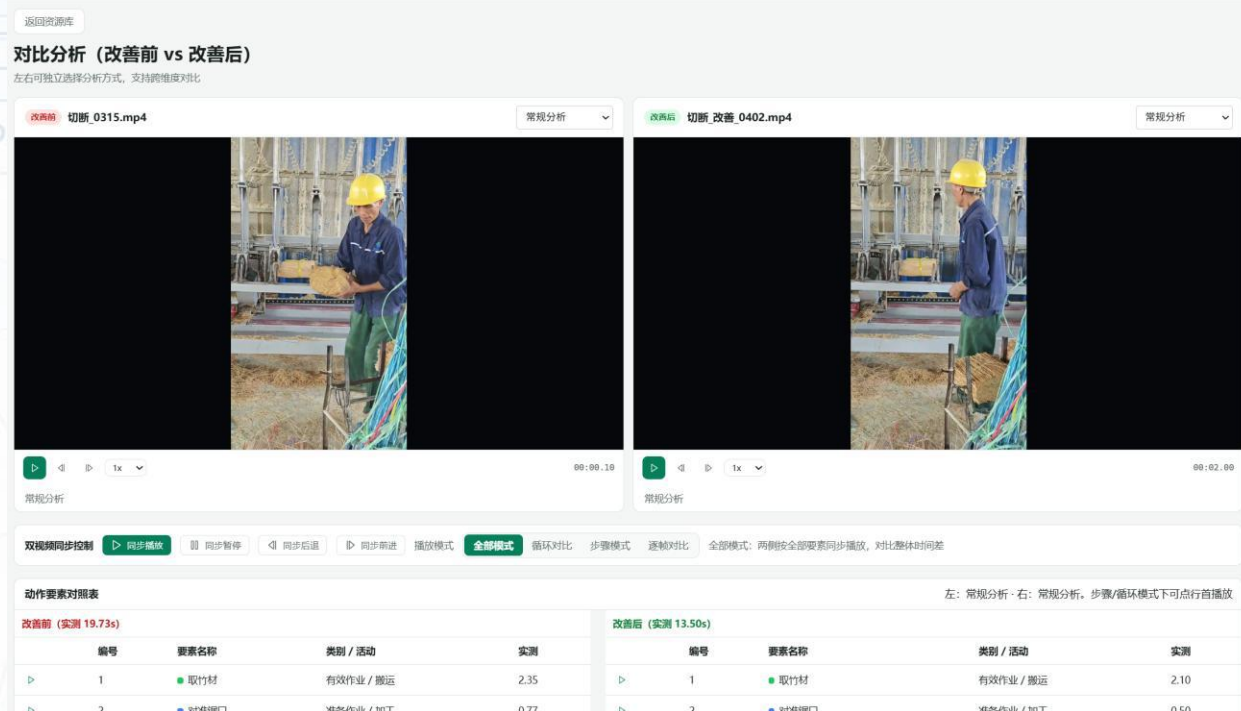


【图 30 线平衡分析页面】

页面上半部分是工序信息卡片表, 列出每道工序包含的作业要素和标准时间; 下半部分是山积图, 每道工序一根柱, 柱内按要素横切堆叠, 虚线是平均工序时间, 高负荷工序被单独标出。工程师可以把要素从瓶颈工序拖到相邻工序, 线平衡率和瓶颈立即重算, 用于推演改善方案, 也可一键重置。页面给出线平衡率、瓶颈工序、平均工序时间和工序数, 节拍、人数和单位小时产量依赖需求量、班制和出勤等输入, 留给工程师结合生产计划另行确定。

(七) 改善前后对比

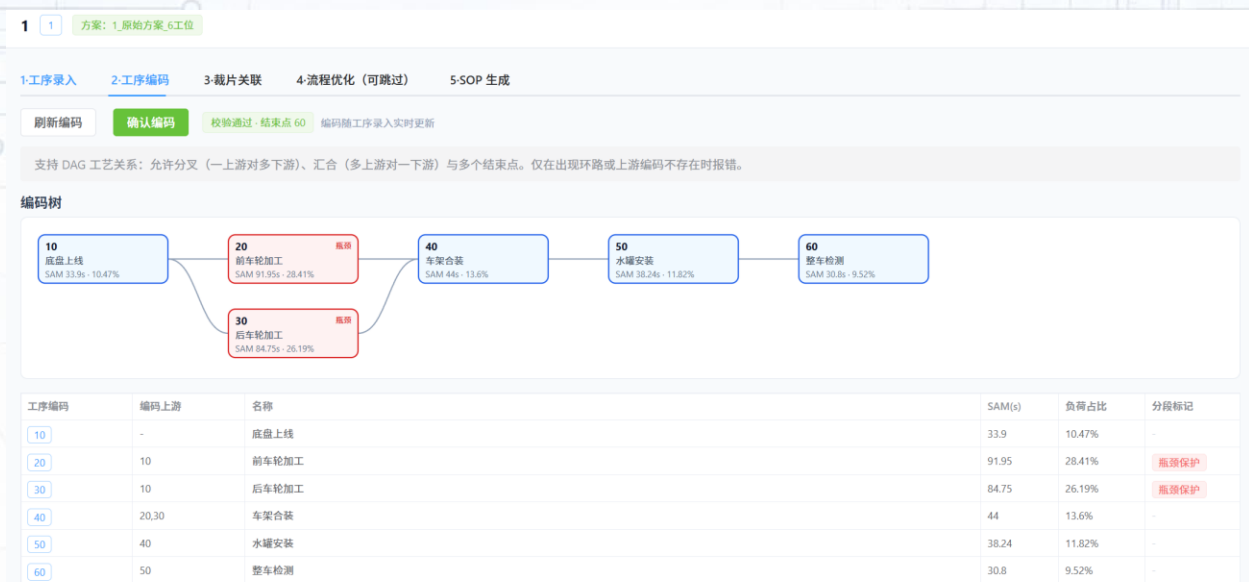
在资源库中先后选中两个视频即进入对比分析, 点选顺序就是改善前和改善后。两段录像左右并排播放, 可按全部同步、循环同步、按步骤同步或逐帧同步四种方式对齐, 下方各自显示要素表和实测周期, 哪一步被删掉、被缩短一眼可见。两边按同一口径计算, 智能体可据此写出周期和各类作业时间变化的结论。



【图 31 改善前后对比分析】

(八) SOP 编制与导出

标准作业指导书是改善成果落地的载体。SOP 编制同样以文件夹为输入, 按工序录入、工序编码、裁片关联、流程优化和 SOP 生成五步进行。系统按工序顺序生成工序编码 10、20、30, 默认上游是前一道工序, 也可以手工指定多个上游, 支持一道工序分叉到多道下游、多道上游汇合到一道工序的工艺关系。编码树自动检查缺编码、重复编码、上游不存在和循环引用, 每个节点标出 SAM 和负荷占比, 瓶颈工序以红色突出。



【图 32 SOP 工序编码树】

每道工序的标准时间和 SAM 从要素表汇总, 平均评比系数和平均宽放率按有效时间加权, 保证工序汇总与要素计算一致。生成页面按标准作业指导书的格式排版, 表头包括产品名称、工序名称、工序排号、标准工时和标准产能, 作业步骤取自要素名称, 作业照片从视频截帧, 不必去现场补拍, 物料和设备清单可从 Excel 导入。

SOP 编制

SOP 生成

按 Excel 模板补齐或修改, 产品名称只准当前文件命名

导出 Excel 选用视频帧

标准作业指导书 (SOP)

文件编号: 编制日期: 页数: 1 版本号: A

产品名称: 发动机装配线 工序名称: 02_定子安装 标准工时: 18.6s 标准产能/H: 194

工序排号: 20 作业类型: 人员配置: 1

序号	材料编号	材料名称	材料规格	数量
1	M12-BOLT	六角螺栓	M12x40	1
2	M12-NUT	六角螺母	M12	1

检查工序 + 添加一行

本工序作业

1. 操作机壳组件 删除
2. 持连接器对位 删除
3. 在定子处操作 删除
4. 整理细金属销 删除
5. 送细杆至机台 删除
6. 持小件准备装配 删除
7. 整理小件 删除
8. 移动小件至定子 删除

【图 33 SOP 生成页面】

导出的 Excel 作业指导书包含工序编码、作业步骤、标准时间、SAM 和视频截帧, 物料、设备和审批栏留给现场填写签字。智能体也可以在对话中直接为一道工序或整条产线导出 SOP。从录像到 AI 初稿、

人工复核、标准时间计算、线平衡推演，再到带截帧的作业指导书，工时分析模块把测定、分析、改善、标准化串在一条数据链上，每个数字都能追溯到某段录像的某个版本。

六、接口与数据

知绘 IE 的智能体工作台和三个业务模块在同一台电脑上作为独立的服务运行。这带来两个设计问题：各部分怎样通信，才能同时支持对话驱动和页面操作；数据怎样组织，才能让模型拿到的数字和页面上的数字完全一致，并在多次修改、AI 写入之后仍然可以追溯。本章先说明通信方式和核心接口，再介绍核心数据实体。

（一）通信方式

通信分为三段。第一段在用户界面和智能体之间：用户的输入和追问选择以普通的 HTTP 请求提交，智能体的回复通过服务器推送事件流（SSE）逐字推回界面。事件流每 30 秒发送一次心跳，断线重连时服务端先补发当前正在生成的消息快照，再继续推送增量，回复内容不丢失也不重复。对话本质上是一问一长答，单向推送已经够用，实现和排错都比双向长连接简单。

第二段在智能体和业务模块之间。三个模块各自是一个本机 HTTP 服务，请求和响应都是 JSON。智能体不直接操作数据库，每一个业务工具最终都落到模块接口上，与用户在页面上点按钮走的是同一套接口。业务规则因此只在模块后端实现一次，不会出现“对话里能做、页面上不允许”的分叉。

第三段在模型和智能体之间。模型看到的不是 HTTP 地址，而是 25 个带名称、说明和参数定义的工具。模型提出调用后，智能体先做参数校验和安全检查，再翻译成接口请求，把结果整理成文字和卡片交还给模型。接口地址、登录令牌、分块上传这些细节都封装在工具内部，三个模块风格不一的返回格式也由统一的服务客户端做了适配。

耗时较长的操作采用“提交任务、按任务号查询进度”的异步方式。AI 自动分析提交后立即返回任务号，工具每 2 秒查询一次进度，任务依次经过抽帧、识别画面、归并要素、写入结果四个阶段，最长等待 15 分钟，用户中途可以取消。按模板出图通常在秒级完成，就直接同步返回。作业视频按 4 MB 分块上传，完成时校验文件大小和 SHA-256 摘要，重复导入会被识别出来。

（二）核心接口

下表列出智能体最常调用的 13 个业务接口，各模块的接口统一带有版本前缀，表中省略。

【表 7 智能体常用的核心业务接口】

模块	方法	路径	用途
知识库	POST	/retrieve	混合检索，返回带来源的证据段落，不调用大模型
知识库	GET	/wiki/pages/{slug}	读取完整页面，页面改名后自动跟随到新地址
知识库	POST	/ingestion/upload	上传待入库资料，先做隐私扫描
绘图	GET	/templates/{id}	读取模板的参数定义、示例和填写规则
绘图	POST	/charts/generate	按模板和参数生成图片
绘图	POST	/charts	把生成的图保存为可再编辑的图表记录
绘图	GET	/charts/{id}/export-excel	方法研究类图表导出 Excel
工时	POST	/videos/uploads	发起视频分块上传，完成时校验大小与摘要
工时	POST	/auto-analysis	提交 AI 自动分析任务，立即返回任务号
工时	GET	/auto-analysis/{job_id}	查询任务所处阶段和进度
工时	GET、 PUT	/analyses/{id}	读取分析记录；带版本号保存为新版本
工时	GET	/insights/balance	按产线文件夹计算线平衡率和瓶颈工序
工时	GET	/insights/sop/export	按工序顺序导出带视频截帧的作业指导书

工时模块的汇总接口是专门为智能体设计的。页面上的标准时间、线平衡率由前端按公式计算，汇总接口在后端用完全相同的公式（包括银行家舍入规则）再算一遍，两侧结果有专门的测试比对。智能体报出的数字全部来自这组接口，不会因为模型心算出错而与页面不一致。

错误处理遵循统一约定：参数不合法返回 400 或 422 并指明哪一项不符合要求，资源不存在返回 404，与现有数据冲突返回 409（如重复导入同一段视频、已有正在运行的分析任务、尚未配置视觉模型），模型未配置或模块尚未就绪返回 503。错误信息一律写成用户能看懂的提示，智能体可以直接转述或据此追问。隐私扫描发现身份证号、密钥等敏感内容时，接口直接拒绝并返回打码后的命中片段，而不是悄悄入库。

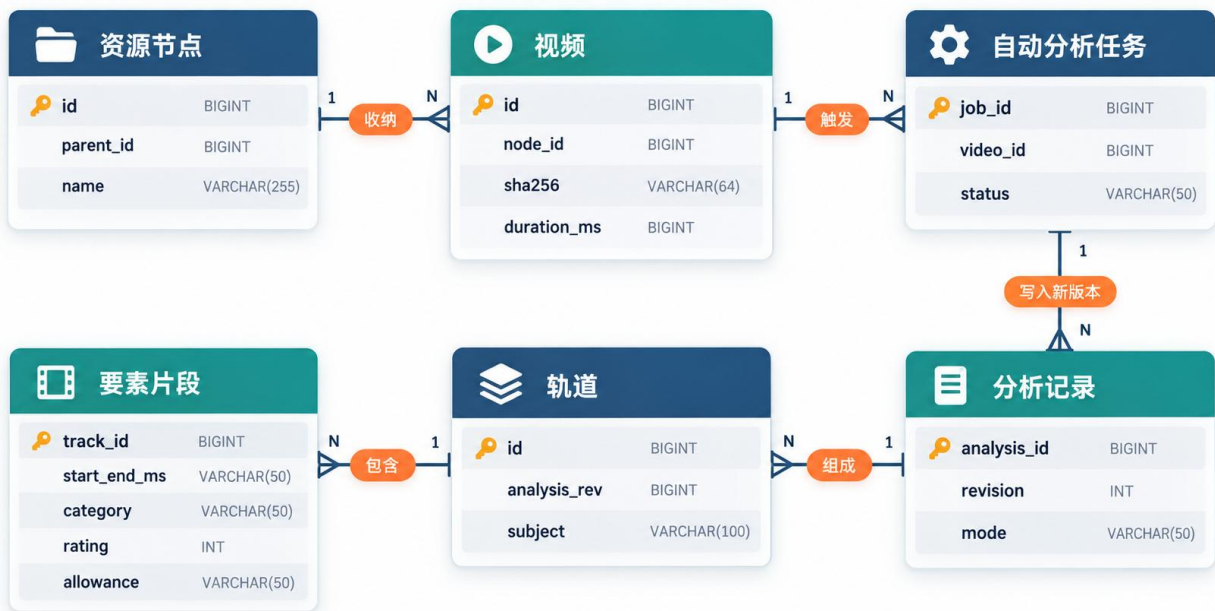
版本冲突是工时模块最重要的一条约定。分析记录每次保存都必须携带当前版本号，所依据的版本已经不是最新时返回 409。它针对的场景是工程师正在页面上调整要素，智能体按另一句指令也修改了同一

份分析，后写入的一方如果直接覆盖，前一方的修改就会无声丢失。有了版本检查，冲突的一方收到明确提示，刷新后在最新版本上继续修改。

(三) 数据模型

三个业务模块各自持有一个单文件的 SQLite 数据库，互不读取对方的数据，模块之间的关联只通过智能体调用接口完成，任何一个模块调整数据结构都不会波及其他模块。全部数据按模块分目录保存在本机的用户数据目录下，可以整体复制备份。

工时分析的数据围绕“产线、视频、分析、要素”这条主线组织。一段视频可以有多份分析记录，同一份分析的所有版本共用一个分析编号，每次保存新增一个版本号加一的完整快照。每个版本下有若干轨道，对应作业者、左手、右手或联合作业中的多名作业者，五种分析方式的区别只在于轨道的组合。轨道下是要素片段，动素代码、模特法分析式和次数都放在同一张片段表中，不用的字段留空。

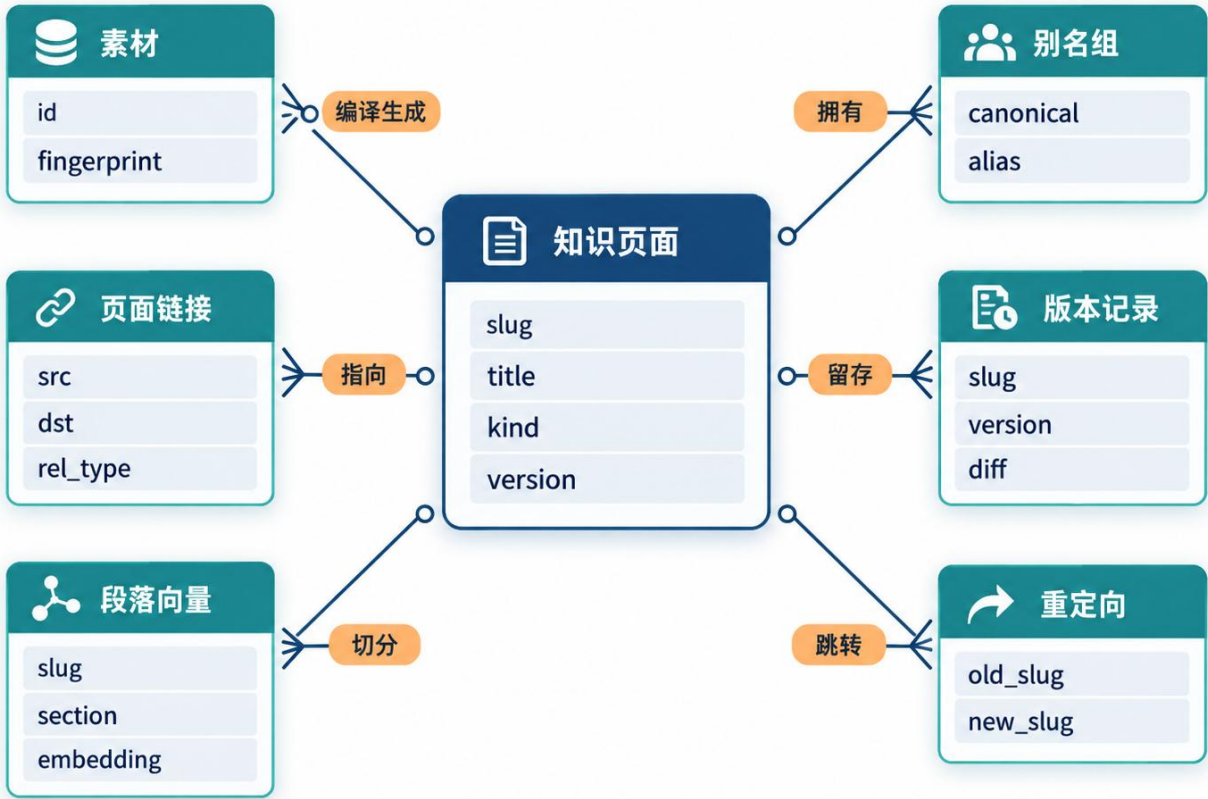


【图 34 工时分析数据关系】

这里有两处取舍。一是版本采用整份快照而不是逐字段记录变更，一份分析通常只有几十个要素，多占的空间很小，换来的是任意历史版本都能直接读出完整状态，改善前后对比也不需要回放修改过程。二是五种分析方式共用一张宽表，切换分析方式时数据结构不变，页面、智能体工具和自动分析流水线只需维护一套校验规则；评比系数限定在 0.01 到 3.00、宽放率限定在 0 到 1、结束时间必须晚于开始时间，这些约束在数据库层面就有检查。

知识库的数据以 Markdown 文件为真源。每个知识页面是一个文本文件，文件头部记录标题、类型、别名、来源资料、版本号等信息，正文用双链语法指向其他页面。数据库只是这些文件的镜像索引，可以随时整体重建：链接表的 31,991 条链接是知识图谱的数据来源，重定向表保证页面改名或删除后历史回答

中的引用不会失效，版本表记录每次修改的差异和触发原因。检索用的向量单独存放在本机向量库中，每个段落一条 1024 维向量，共 24,207 条。这样企业积累的工艺知识以普通文本保存，可以用任何编辑器打开、纳入版本管理，不会被锁在某个数据库格式里。



【图 35 知识库数据关系】

绘图模块的核心是模板和图表记录。模板由启动时扫描模板包同步而来，新增模板只需放入一个模板包。图表记录按引擎分别存储内容：matplotlib 类保存参数和预览图，Vue 类保存表单数据，draw.io 类保存图形的 XML。

下表汇总三块中最核心的 8 个实体。

【表 8 核心数据实体】

实体	模块	关键字段	设计要点
资源节点	工时	名称、上级节点	自引用的文件夹树，文件夹即产线
视频	工时	所属节点、文件名、时长、大小、内容摘要	摘要唯一，防止重复导入
分析记录	工时	分析编号、版本号、分析方式、所属视频	编号加版本号唯一，每次保存追加完整快照

实体	模块	关键字段	设计要点
要素片段	工时	所属轨道、起止毫秒、类别、要素名称、动素代码、MOD 分析式、评比系数、宽放率	五种分析方式共用一张表，约束在库内校验
自动分析任务	工时	视频、分析方式、状态、阶段、进度、结果版本	同一视频同时只允许一个运行中的任务
知识页面	知识库	页面标识、标题、类型、版本、来源资料	文件为真源，数据库为可重建的镜像
页面链接	知识库	源页面、目标页面、链接类型、出现次数、置信度	知识图谱的边，同一对页面同一类型只记一条
图表记录	绘图	标题、所用模板、引擎、图表内容	保存参数而非仅保存图片，可再次编辑

智能体的对话会话以逐行追加的文本文件保存，每条消息记录上一条消息的编号，整份会话构成一棵树，从中间改问另一个问题会形成新的分支，意外断电时最多丢失最后一行。模型配置集中在一份配置文件中，四个服务共用，其中的密钥加密存放。

七、部署、安全与测试

知绘 IE 面向的是制造企业的 IE 工程师，他们的电脑往往由企业统一管理，不一定能自由联网安装软件，录下的作业视频和工艺资料也属于企业内部数据。因此这一章要回答三个问题：软件怎样做到装上就能用；一个能读写文件、执行命令的智能体，怎样保证不破坏系统、不泄露数据；以及凭什么相信它按设计工作。

（一）离线部署

知绘 IE 以一个 Windows 安装包分发，安装时可以选择安装目录，并创建桌面和开始菜单快捷方式。安装包内自带完整的离线运行时，解包后约 1.09 GB，包括嵌入版 Python 解释器、三个业务模块各自的依赖库、离线版 draw.io，以及供智能体在本机生成 Word、Excel 和做数据分析用的办公类库。办公类库占比最大，约 366 MB；知识库、绘图、工时三个模块的依赖分别约 265 MB、208 MB 和 168 MB；draw.io 约 65 MB，Python 解释器本身约 22 MB。各模块的依赖分目录安装，版本互不干扰，办公类库全部固定版本，保证在不同电脑上生成的文档一致。用户安装后不需要再联网下载任何组件，也不需要预先安装 Python 或数据库，这对内网环境和不熟悉开发工具的工程师都很重要。

离线运行时解包后约 1.09 GB

六部分合计 1,094 MB；条长按体积比例绘制



各模块依赖分目录安装，版本互不干扰；安装后无需再联网下载组件，也无需预装 Python 或数据库

【图 36 离线运行时组成】

用户双击图标后，桌面程序先检查是否已有实例在运行，避免重复启动；然后检查运行时是否已经解包且版本一致，首次启动或升级后才解包。解包先写入临时目录，校验完整后再替换，旧版本保留一份，即使解包中途断电也不会损坏已有的运行时。接下来依次拉起知识库、绘图、工时三个后端服务，每个服务启动后都做健康检查，确认就绪再进行下一步；随后启动各模块的页面服务，自动完成模块登录，最后启动智能体工作台并打开主窗口。某个模块没能正常启动时，程序会弹出提示，说明是哪个模块没有就绪。启动所需的端口如果被其他程序占用，程序会直接报错，而不是悄悄换一个端口，这样问题能被及时发现。用户关闭窗口时，桌面程序会结束它启动的全部子进程，不会在后台残留服务。

所有服务都只监听本机地址，局域网中的其他电脑无法访问。数据保存在当前用户的数据目录下，按模块分目录存放，备份时整体复制即可。

(二) 安全与隐私

智能体能在本机读写文件、执行 PowerShell 和 Python 脚本，这是它能替工程师写报告、整理数据的前提，也是最需要防范的地方。模型可能理解错指令，网页或资料中也可能夹带诱导模型执行危险操作的文字。知绘 IE 的做法是在每一次工具调用真正执行之前做三道检查：路径守卫、危险命令规则和权限模式确认。

路径守卫把本机路径分为三类。第一类是软件自己的数据目录，里面有会话记录、模型配置、密钥文件和各模块数据库，智能体既不能写也不能读。连读取都禁止，是因为一旦智能体能读到密钥文件，资料中的一段恶意文字就可能诱导它把密钥写进回答里。第二类是软件安装目录，允许读取，禁止修改。第三类是 Windows 系统目录和程序目录，同样禁止修改。路径在比较之前统一换算成标准写法，模型换用 Linux 风格路径、相对路径或其他变体都绕不过去；执行命令时，守卫还会检查命令文本中是否出现受保护目录。

危险命令规则针对的是即使路径合法也不应执行的操作，例如格式化磁盘、关机重启、修改启动配置、删除注册表、关闭安全防护、删除系统备份、添加系统账户，以及删除用户目录或根目录。一条命令行会先按分号、换行和逻辑连接符拆成单独的语句再逐句匹配，既防止把危险操作藏在长命令中间，也避免把两句无关的语句拼在一起造成误判。被拦截时，智能体会收到明确的原因，并被要求换一种不破坏系统和数据的做法或请用户自己执行，系统提示中也规定被拦截后不得换一种写法绕过。

权限模式决定智能体在多大程度上需要征得用户同意，用户可以在对话中随时切换。“每次询问”模式下，检索知识库、查询分析结果这类只读操作直接执行，写文件、执行命令等操作每次都会弹出确认框，显示要调用的工具和参数预览。“自动”是默认模式，操作自动执行，只依靠守卫和危险命令规则把关，适合日常使用。“完全放开”模式面向熟悉软件的用户，额外放开安装目录的修改和少数开发类命令，但数据目录和系统目录在任何模式下都受保护。在没有界面可以确认的场合，需要确认的操作一律按拒绝处理。

模型密钥在本机加密后保存，智能体和各模块后端共用同一份配置和同一把加密密钥，不需要在多处保存明文。设置界面只显示密钥的首尾几位，接口返回的错误信息在送往界面前会去掉密钥。密钥文件位于受保护的数据目录中，智能体的文件工具读不到。智能体工作台还会校验每个请求的来源，只接受本机同源请求，防止用户在浏览器中打开的外部网页向本机服务发请求、驱动智能体。

知绘 IE 唯一的对外通道是用户自己配置的模型接口。下表列出各类用途会发出和不会发出的内容。

【表 9 发往模型接口的数据】

用途	发出的内容	不发出的内容
对话	对话文字、工具返回的文字结果	视频文件、数据库文件
向量	知识库段落文字、检索问题	页面以外的本机文件
视觉	抽帧得到的压缩图片，短边不超过 480 像素， 每批 12 帧	原始视频
资料入库	待入库资料的正文	隐私扫描拒绝的资料
联网检索	检索关键词，仅在用户填写检索服务密钥后启用	本机文件内容

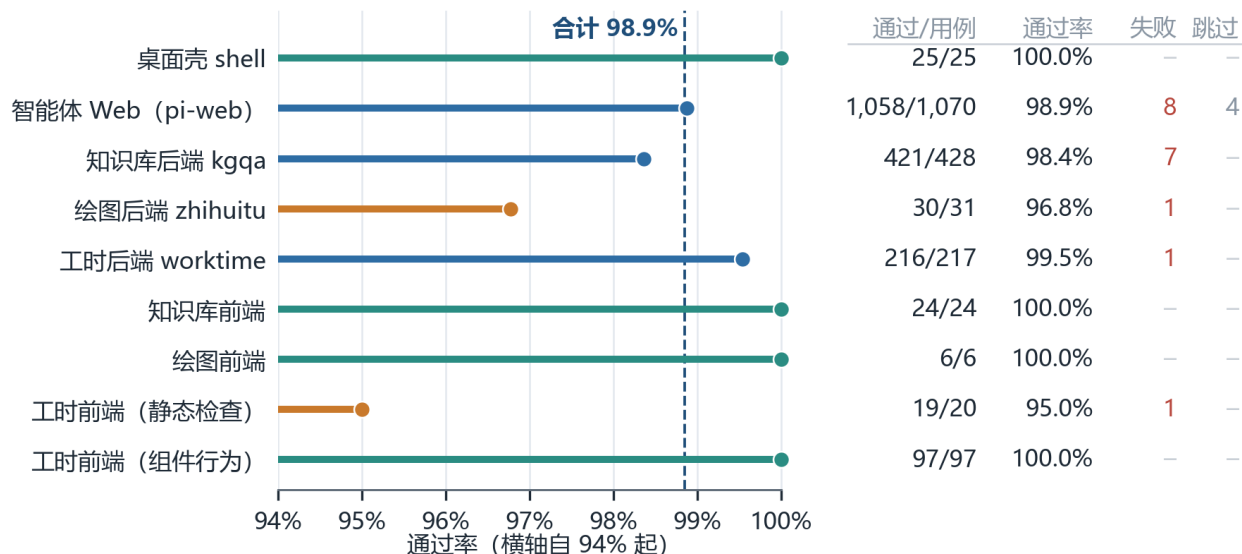
资料入库前会先做隐私扫描，发现身份证号、密钥或带令牌的网址时拒绝入库，发现手机号、邮箱、银行卡号时给出警告，并把命中的片段打码后展示给用户。智能体底座的遥测功能在知绘 IE 中默认关闭。对数据不出厂有硬性要求的企业，可以把对话、向量、视觉三类模型都配置为本机 Ollama 部署的开源模型，这时整个系统没有任何外部网络流量，视频、资料和分析结果全部留在这台电脑上。

(三) 自动化测试

知绘 IE 的自动化测试共 1,918 项，覆盖桌面程序、智能体工作台、三个业务模块的后端和前端，通过 1,896 项，通过率 98.9%。测试在隔离环境中运行：用户目录、临时目录、各模块的数据库、知识库目录和视频目录全部指向临时的沙箱位置，网络代理指向不可达的地址，模型密钥置空，测试结束后核对真实数据库和知识库文件均未发生任何变化。需要真实模型接口的用例由测试自身跳过。

各子项目自动化测试通过率：1,896 / 1,918 项通过 (98.9%)

自研代码 9 个测试套件，不含上游 Pi 内核；失败项均不涉及业务逻辑（测试滞后、Windows 平台差异、中文文案检查）



【图 37 各子项目自动化测试通过率】

用例数量最多的是智能体工作台，共 1,070 项，通过 1,058 项，其中知绘 IE 新增的集成层 125 项全部通过，内容包括守卫拦截删除数据目录的命令、模型配置的加密保存与迁移、三个业务模块的工具适配、追问卡片等。知识库后端 428 项，通过 421 项，覆盖页面解析、链接与重定向、混合检索、隐私扫描、知识图谱构建，以及用模拟模型跑通的完整入库流水线。工时后端 217 项，通过 216 项，覆盖用模拟视觉模型跑通的自动分析流水线、时间轴、线平衡和智能体使用的汇总指标。工时前端的 97 项组件行为测试和桌面程序的 25 项测试全部通过，后者检查模块清单不泄露密钥、端口冲突时立即报错、打包时排除配置文件和数据库等。绘图后端 31 项通过 30 项，三个模块前端的静态检查共 50 项通过 49 项。标准时间、线平衡率、MOD 换算、银行家舍入、版本冲突检查、检索融合排序和实体判重这些直接决定产出数字的核心计算都有专门用例，页面与汇总接口两侧的计算结果由比对测试逐项校验，保证同一个数字在页面、对话和导出文件中处处一致。

八、应用案例

本章用三个案例说明知绘 IE 在实际工作中替用户做了什么。三类用户关心的事情不同：学生要“学得会、做得完”，在有限课时内走完方法研究、作业测定和线平衡的全过程；没有专职 IE 的中小企业要“没

人也能做”，由生产主管拿录像就能摸清现状、找到瓶颈；有 IE 团队的企业要“少做重复活”，把逐帧测时、画图、排表交给软件，工程师把精力放在判断上。每个案例都按“用知绘 IE 做了什么、得到什么产出、传统做法要花多少功夫”来写。文中图表除现场照片外均由知绘 IE 生成或导出，传统做法的耗时是按常规手工流程的估算。企业名称做脱敏处理，数据来自实验小组和企业提供的资料以及知绘 IE 中保存的分析记录。

（一）IE 课程实验：学生小组用知绘 IE 完成消防车装配线实验

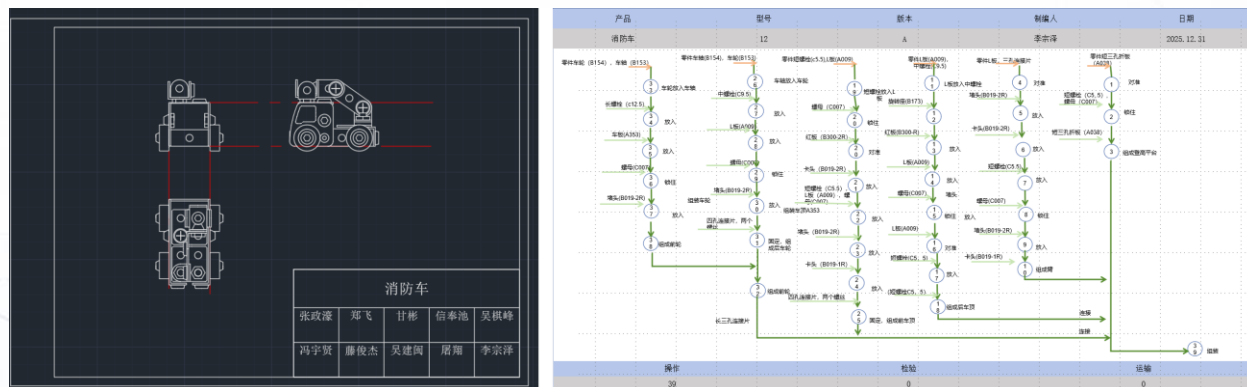
1. 实验概况与痛点

本案例来自高校《基础工业工程》课程实验。8 人小组在实验室搭建了一条消防车金属拼装玩具的模拟装配线，产品由 16 类、55 个零件组成，装配顺序为前车轮、后车轮、前车顶、后车顶、臂和登高平台，最后总拼装。全线划为 8 个工位，每人负责一个，全部手工装配。

按实验指导书，小组要依次完成程序分析、作业分析、动作分析、秒表测时与标准工时、线平衡、学习曲线和 5S，每一项都要交图表。过去这些图表靠手绘或在 Excel 里逐格画，8 个工位的双手作业图、动素分析表和测时表加起来是一周以上的工作量（估算），学生的大部分时间花在画图和抄数上，留给分析和改善的时间很少。这次小组全程使用知绘 IE，各环节的做法如下。

2. 程序分析：描述装配顺序，模板直接出图

组员把 BOM 和装配顺序告诉智能体，请它画工艺程序图。智能体调用绘图模块的工艺程序图模板，把 55 个零件按前车轮、后车轮、前车顶等部件组织成参数，资料中没写清的地方先用追问卡片问清楚，再出图。图以卡片形式回到对话中，需要调整时直接在 draw.io 中修改。流程程序图也这样生成，操作、检验、搬运、等待、储存的次数随图统计，并可导出带格式的 Excel。手绘一张包含 55 个零件的工艺程序图，加上反复修改，通常要 2~3 h（估算）；在知绘 IE 中几分钟出图，核对完不到半小时。



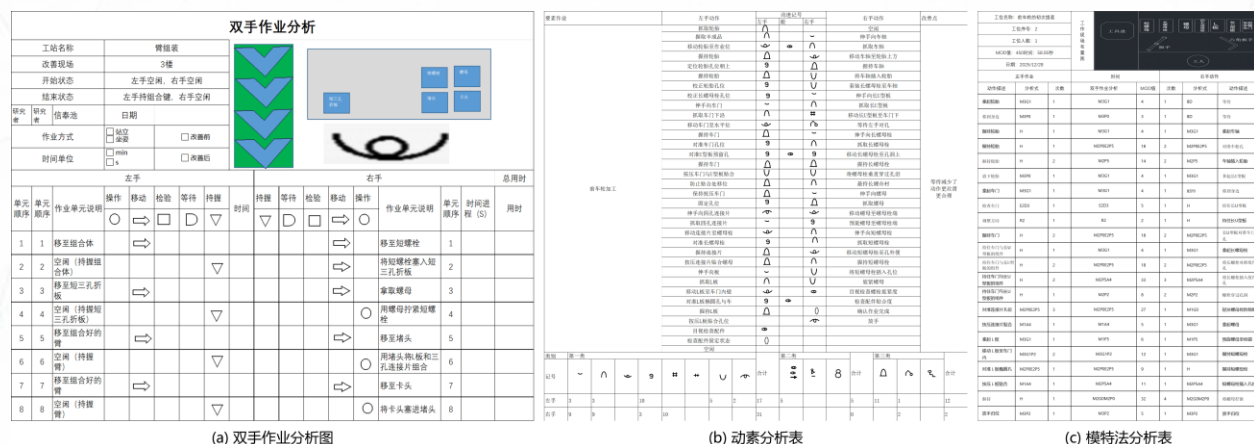
【图 38 消防车总装图与知绘 IE 生成的工艺程序图】

3. 作业分析：录像交给 AI，四种方式分别分析

小组用手机固定机位，为 8 个工位各录制 10 个完整循环，在资源库中新建“消防车装配线”文件夹，按 S1 到 S8 放入。与实验报告的安排一致，每个工位的录像都分别做常规分析、动素分析、双手分析和模

特法分析，在工时分析工作台切换分析方式即可，同一段录像不必重新导入。AI 给出初稿后，由负责该工位的组员复核边界和类别，调整主要集中在螺母对准与拧紧的切换处。

四种分析各对应报告中的一项内容：常规分析把循环拆成作业要素，区分有效、辅助和浪费时间，是测时和线平衡的基础；动素分析按左手、右手标注伸手、握取、移物、定位、装配、持住等动素，对应动作分析表；双手分析让左右手在同一时刻切分、逐行对齐，得到教材格式的双手作业图；模特法按 MOD 给出理论时间，用来与观测时间对照。四种结果都可以调用绘图插件生成对应的分析图表，并导出 Excel 直接放进实验报告，下图是几张示例。



【图 39 知绘 IE 生成的双手作业分析图、动素分析表和模特法分析表（示例）】

4. 作业测定：不会就问，按公式出标准工时

测时环节最容易出错的是评比和宽放。几位组员不清楚平准化法怎么打分，直接在对话中提问。智能体检索知识库，回答附编号证据，点开即可看到教材原文，不同教材的定义并列显示：

[E1] 平准化法从熟练、努力、工作环境和一致性四个因素评定作业者的表现，每个因素分为六个等级并对应系数，四项系数之和加 1 即为评比系数。（知识库：基础工业工程）

[E2] 正常时间等于观测时间乘以评比系数，标准时间等于正常时间乘以（1 + 宽放率）。（知识库：基础工业工程）

组员据此填入各工位的评比系数和 15% 的宽放率，知绘 IE 以文件夹为单位按“标准工时 = 观测 × 评比系数 × (1 + 宽放率)”算出 8 个工位的标准工时，每个数字都能点回对应工位的要素表和录像片段，结果如下表。手工汇总 8 份测时表再逐个计算约需半天。

【表 10 8 个工位的测定结果】

工位	作业内容	观测均值 /s	评比系数	标准工时 /s
S1	前车轮组装, 检查螺丝紧固	37.81	1.25	54.35
S2	前车轮加工, 检查螺丝紧固	20.00	1.15	26.45
S3	前车顶组装, 四孔连接片与后轮固定	68.40	1.15	90.46
S4	部分车顶与左后轮组装	70.40	1.00	80.96
S5	右后车轮与后车顶组装	64.00	1.00	73.60
S6	后车顶、臂、登高平台连接	58.12	1.02	68.17
S7	前轮、前车顶、后轮连接	57.60	1.04	68.89
S8	整体组装与臂的安装	30.00	1.25	43.13
合计	—	406.33	—	506.01

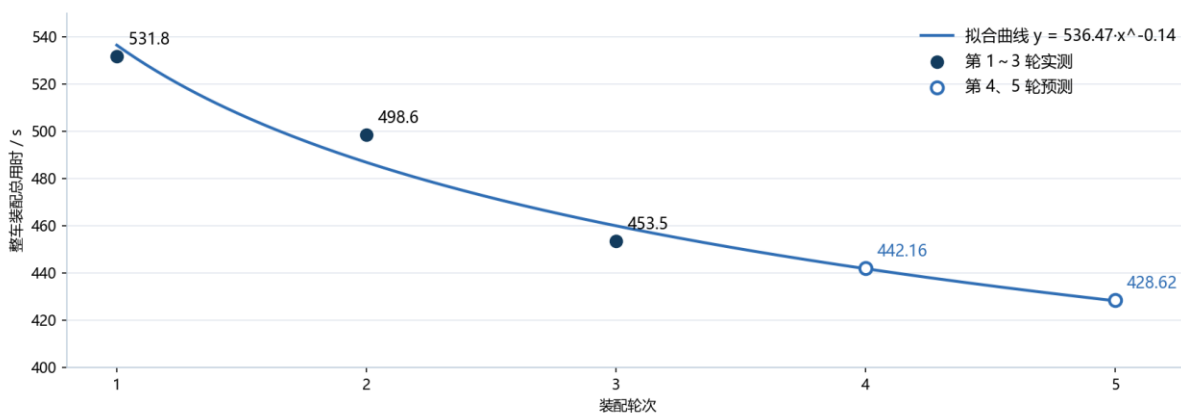
5. 线平衡与学习曲线：拖动要素即可比较方案

按标准工时，瓶颈是 S3 的 90.46 s，线平衡率 69.92%；S3、S4 都在 80 s 以上，S2 只有 26.45 s，最长与最短工位相差 3.4 倍。要素表按动作汇总后，拧螺母和紧固螺栓类要素合计 230.24 s，占全线观测时间 406.33 s 的 56.7%。组员问智能体“S3 是瓶颈，该怎么改”，智能体结合知识库中的动作经济原则和生产线平衡条目给出三个方向：螺母改用套筒扳手，增加定位夹具让持件的手空出来，把 S3、S4 中的两项预装移到负荷最低的 S2，每条都附证据编号。组员在山积图上把预装要素拖到 S2，瓶颈和线平衡率随即重算，几种分配方案几分钟就能比完；手工重算一次全线平衡要十几分钟。

学习曲线环节，组员把前三轮整车装配总用时 531.84 s、498.6 s、453.5 s 交给学习曲线模板，模板做幂函数拟合，得到 $y = 536.47x^{(-0.14)}$ ， $R^2 = 0.934$ ，学习率约 90.8%，并预测第 4、5 轮分别为 442.16 s 和 428.62 s。整车装配总用时是 8 个工位串联完成一台车的实测时间，包含工位间传递，与各工位标准工时之和不是同一口径。在 Excel 中取对数、回归、作图约需半小时（估算），这里一句话出图，后续评价改善效果时可以用这条曲线扣除熟练带来的下降。

整车装配总用时的学习曲线（绘图模板：学习曲线）

前三轮实测 531.84 / 498.6 / 453.5 s, 幂函数拟合 $R^2 = 0.934$, 学习率约 90.8%



【图 40 整车装配总用时的学习曲线（绘图模板：学习曲线）】

6. 标准化：生成作业指导书和 5S 检查表

复核完成的要素表就是标准时间基线。小组用产线 SOP 工具选中“消防车装配线”文件夹，按 S1 到 S8 的顺序生成 8 个工位的作业指导书，工序编码 10 到 80，作业步骤取自要素名称，作业照片从录像截帧，不必再到现场补拍，导出 Excel 后张贴在各工位。5S 环节，组员请智能体依据知识库中的 5S 条目，按整理、整顿、清扫、清洁、素养五项生成工位检查表和评估表，智能体直接输出 Word 文件，组员按实验室情况增删条目后用于各工位评估。手工排版 8 份带照片的作业指导书约需一天。

7. 对学生的价值

各环节的做法与传统做法对比如下表。

【表 11 课程实验各环节的做法对比】

环节	知绘 IE 中的做法	产出	传统做法（估算）
程序分析	描述 BOM 和装配顺序，模板出图，缺数据时追问	工艺程序图、流程程序图	手绘 2~3 h
作业分析	录像导入，AI 分别做常规、动素、双手、模特法分析，组员复核	8 个工位的要素表、动作分析表、双手作业图	逐帧记录加画表，好几天
作业测定	知识库答疑评比与宽放，按公式计算	8 个工位标准工时，合计 506.01 s	人工汇总计算，约半天
线平衡	山积图拖动要素，自动重算	瓶颈 S3、线平衡率 69.92%，方案比较	每次重算十几分钟
学习曲线	输入三轮用时，模板拟合	拟合公式与预测值	Excel 回归作图约半小时
标准化	产线 SOP 工具生成；智能体生成 5S 检查表	8 份带截帧的作业指导书、5S 检查表	排版约一天

画图和抄数的时间省下来后，学生把精力用在看懂录像、讨论改哪里上。遇到不会的概念随时可以问，回答都附教材出处，不会被似是而非的说法带偏。分析记录只追加不覆盖，AI 初稿和组员复核版本保存在同一条版本链上，指导教师可以直接查看每位组员改了哪些边界、为什么改，比批改手工表格更容易发现测时中的问题。

（二）Z 竹业：没有专职 IE 的企业用录像找到瓶颈

1. 企业概况与问题

Z 竹业是一家以毛竹为原料的小型加工企业，2025 年采购毛竹 8,662 t，产品有地板条、竹帘、竹筷和香芯。厂里没有专职工业工程人员，工时和产能长期靠班组长经验估计。企业的问题很直接：没有落到秒的现状基线，看不清工位上多少时间在作业、多少时间在等待；每年 10 月前后旺季总要加班，不知道瓶颈在哪一道工序。请顾问驻场测时诊断，费用和周期都超出这类小厂的承受范围。

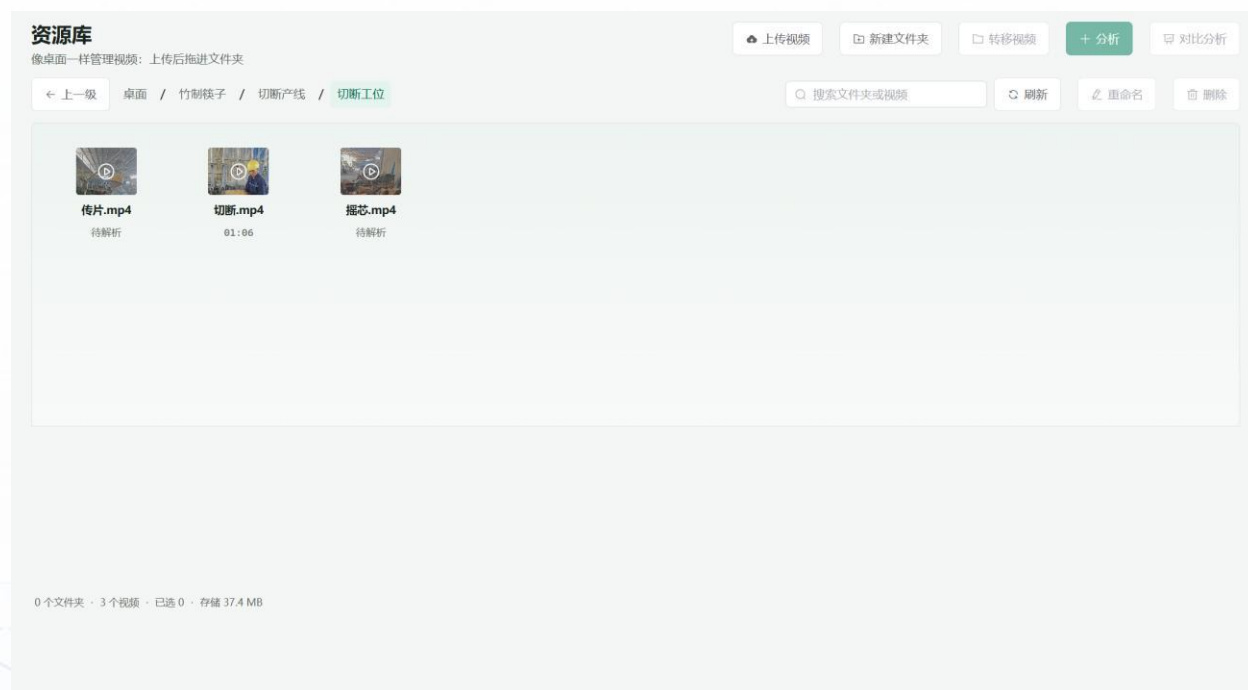
生产主管用手机对采段到摇芯的 7 道主要工序逐一录像，均为 30 fps，时长 11.33 s 到 66.10 s，覆盖竹帘和香芯两条主路径。



【图 41 Z 竹业工序录像代表帧与成品仓库】

2. 录像交给智能体，全线统一分析

生产主管在资源库中新建产线文件夹，把 7 段录像按工序顺序上传，然后在对话中让智能体把这条线的录像都分析一遍。与课程实验一样，每段录像都分别做常规分析、动素分析、双手分析和模特法分析，智能体逐段发起 AI 自动分析，进度在对话中实时显示。

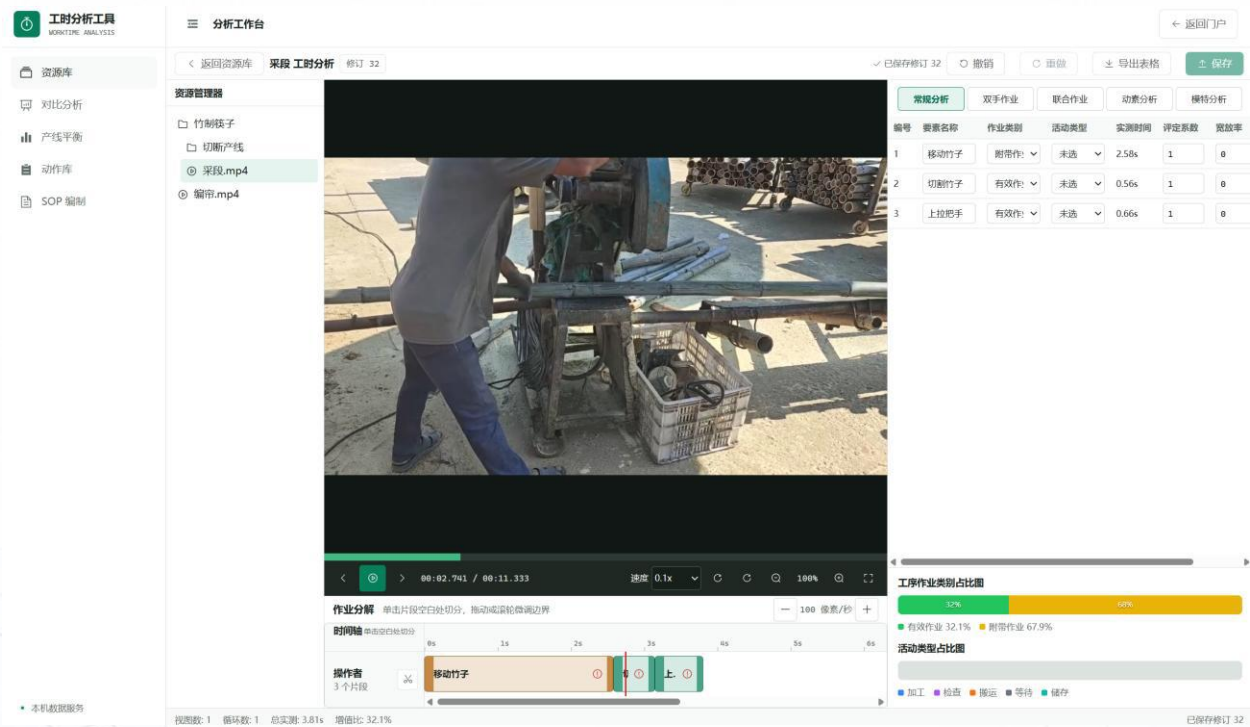


【图 42 产线文件夹与工序视频列表】

【表 12 7 道工序的录像分析概况】

工序	录像 /s	分析方式
采段	11.33	常规分析、动素分析、双手分析、模特法分析
破竹	17.80	常规分析、动素分析、双手分析、模特法分析
抛光拉丝	17.00	常规分析、动素分析、双手分析、模特法分析
切断	66.10	常规分析、动素分析、双手分析、模特法分析
编帘	34.99	常规分析、动素分析、双手分析、模特法分析
磨光	18.73	常规分析、动素分析、双手分析、模特法分析
摇芯	62.49	常规分析、动素分析、双手分析、模特法分析

以编帘为例，34.99 s 的录像常规分析用时 4 分 16 秒得到 10 个要素，动素分析用时 4 分 52 秒得到 25 个带左手、右手和眼部动素代码的要素；采段 11.33 s 的录像用时 80 秒得到 8 个要素。人工逐帧切分这样一段录像要十几分钟到半小时，动素分析更久。



【图 43 编帘工序的要素时间轴（工时分析工作台）】

3. 人工审核：微调即可

AI 分析的结果与现场情况基本相符，生产主管在时间轴上检查要素边界和类别，只对个别边界做了微调，主要是操作员被机器遮挡的片段。没有 IE 背景也不必从零学逐帧切分，只需判断 AI 切得对不对。

复核后的结果显示，几道手工工位的共同问题是人等机器：编帘 34.99 s 的循环中只有 6.25 s 在送料、操作机器和剪竹条，其余 28.74 s 在等编帘机自动编织；采段的浪费占 51.5%，包括切割后持料原地等待和离开工位；破竹取料时弯腰占 21 MOD 中的 17 MOD。按设备台数核算，大破机全厂只有 1 台，是旺季加班的直接原因。

4. 智能体检索知识库，给出改善方案

生产主管把分析结果交给智能体，请它结合知识库提出改善方案。智能体检索了瓶颈工序、人机分离、一人多机作业、ECRS 四原则和动作经济原则等条目，给出四条方案，每条附证据编号，点开即可查看知识库原文。各证据的要点如下。

[E1] 瓶颈工序的节拍决定了整条生产线的最大产出率，改善生产线的首要任务就是识别并消除或缓解瓶颈工序。（知识库：瓶颈工序）

[E2] 设备在自动运行时，人可以离开，去操作另一设备或从事其他工作，即所谓人机分离。（知识库：人机分离）

[E3] 理论最大可操作台数 $N = 1 + M/t$ ，M 为机器自动时间，t 为操作者的手动时间，前提是机器自动时间不小于操作者在其他机器上的手动时间之和。（知识库：一人多机作业）

[E4] 取消是改善的最高境界，经“是否必要”的提问后，不必要的工序、作业和动作予以取消。（知识库：ECRS 四原则）

[E5] 以最低限的疲劳获得最高的效率，在改善动作的同时改善物料布置与工夹具。（知识库：动作经济原则）

【表 13 智能体给出的改善方案】

方案	对象	改善措施	依据	按软件口径推算
方案一	大破	增加 1 台大破机，与原机并行	[E1]	缓解全线唯一约束，需设备投资
方案二	编帘	人机分离，编织等待期间由同一操作员照看第二台编帘机	[E2][E3]	手动 6.25 s，机器自动 23.24 s， 理论上限 $N \approx 4.7$ 台
方案三	采段	取消切割后的持料等待和离开工位，下一根毛竹提前就位	[E4]	循环 11.33 s 缩短到 9.00 s，减少 20.6%
方案四	破竹	料堆垫高到腰部，取消弯腰取料	[E5]	取料 21 MOD 降到 4 MOD，每次少 2.19 s

只有方案一需要投资，其余三条只需调整作业方法或现场布置。方案二的台数上限没有计入两台机器之间的走动，编帘机两台一组相邻布置，智能体建议先按 1 人 2 机试点。每条建议都能追到知识库原文和本厂的要素数据，没有 IE 背景的管理者也看得懂它从哪里来。

5. 方案落地

企业决定方案一走设备采购流程，方案二到方案四先在班组内试点。试点工位按新方法重新录像，放回原产线文件夹，按同一流程分析和复核，前后两版记录同时保留。生产主管在对比页面把改善前后的两段录像并排同步播放，逐个要素核对等待和离开工位是否消失、弯腰取料是否取消。新方法稳定后，用新版要素表在产线 SOP 工具中重新生成这几个工位的作业指导书，作业照片直接取自新录像，下发到班组。

6. 与传统做法的对比

【表 14 顾问诊断与知绘 IE 的做法对比】

环节	请顾问或人工完成（估算）	用知绘 IE 完成
现状测时	顾问驻场秒表测时，数天	手机录像上传，AI 几分钟出初稿
要素拆分	逐帧切分，每段十几分钟到半小时	AI 出初稿，人工微调
问题量化	手工汇总、作图	时间轴与要素表按类别自动统计
改善方案	依赖顾问经验	智能体检索知识库，每条附原文出处
效果确认	再次驻场测时	同一文件夹重新录像，对比页面并排对照
作业标准化	手工编写，现场补拍照片	按文件夹生成带截帧的作业指导书

这家企业没有增加 IE 人员，生产主管用一部手机和知绘 IE 完成了现状测定、瓶颈识别和改善方案制定，分析流程和要素表都留在软件里，下一轮改善可以直接沿用。

(三) X 复材：IE 团队把重复工作交给知绘 IE

1. 企业概况与问题

X 复材生产碳纤维自行车车架，IC114-1 车架线单班 8 h，订单量约 60 件/班，现状单班产出约 17 件。车架依次经过取长、机加工、清洗烘烤、CNC、预组配、粘着、精修 7 道工序，共 21 个作业要素；机加工要按车型更换头管雕刻模具，一次换模停机 960 s。企业的 IE 小组计划用价值流图和鱼骨图定位浪费，再用 SMED、0-1 整数规划和 5S 分别解决换模、负荷不均和现场管理问题，用 FlexSim 仿真验证。方法都很清楚，大量时间却耗在画图、拆录像、整理表格这些重复工作上。小组把这些重复工作交给知绘 IE，自己负责判断和建模。

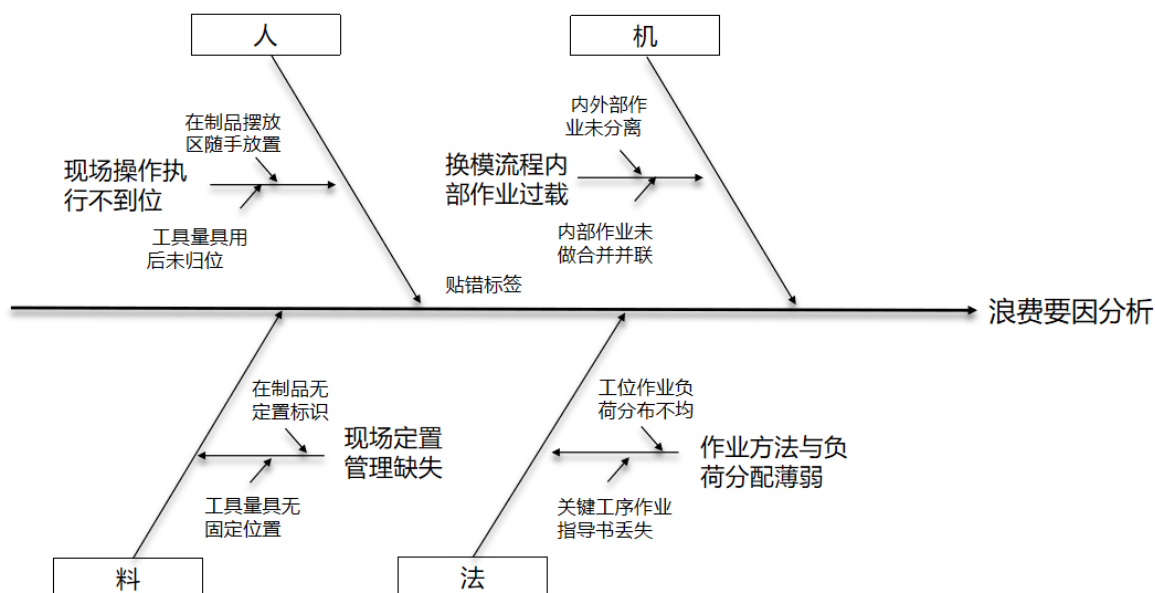
2. 数据交给 AI，生成现状图

小组把各工序的作业要素时间整理成表，连同人数、工序间在制品、批量和班制，一起粘贴到对话中，请智能体画现状价值流图。7 道工序的数据如下表。

【表 15 车架线 7 道工序的现状数据】

工序	作业要素 (标准时间 /s)	工序时间 /s
取长	水喷砂 57.2、取长 181.5	238.7
机加工	雕刻 1056、铣弧 180.4、铰孔 283.8、钻孔 110	1630.2
清洗烘烤	超声 79.2、烘烤 237.6、削毛边 40.7、QC 42.9	400.4
CNC	CNC 793.1	793.1
预组配	预组配 237.6	237.6
粘着	填胶 298.1、点加热 346.5、磨胶 148.5、补纱 198、烘烤 457.6	1448.7
精修	磨胶 209、剖沟 151.8、粘配件 128.7、QC 量测 122.1	611.6
合计	21 个作业要素	5360.3

智能体调用价值流程图模板，把这些数据组织成参数，客户、供应商和信息流一并画出，生成的图可以在 draw.io 中继续修改。手工在 Visio 中画一张 7 道工序的价值流图，排版加核对要半天左右（估算）；在知绘 IE 中一次对话出图，数据有改动时说一句话就能重画。



【图 46 车架线浪费要因鱼骨图】

3. 换模录像拆成 35 个要素，工程师专注判断

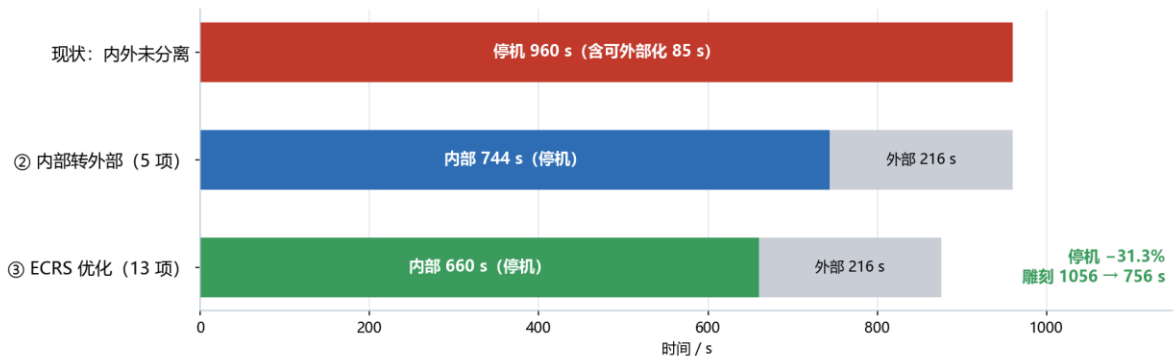
960 s 的头管雕刻换模录像交给 AI 常规分析，最多抽 240 帧，约每 4 s 一帧，换模要素平均约 27 s，抽帧间隔足够区分动作。AI 初稿拆出 35 个要素，工程师复核边界后，在要素表上逐项判断内部作业和外部作业。这一步需要对设备和工艺的理解，仍由工程师完成。

现状 35 项中 30 项共 875 s 是内部作业，占 91.1%，只有取下手套、拿取表格、填写首件确认表、放回表格和戴上手套 5 项共 85 s 是外部作业。工程师把搬运至模具台、找寻新模具、搬运回机台、准备扳手和观察点位 5 项共 131 s 转为外部作业，内部作业降到 744 s；再按 ECRS 对剩余内部作业逐项提问，取消 4 项、合并 2 项、重排 2 项、简化 5 项，例如用镜子观察点位省去拆卸步骤、合并手套动作、首件确认表与换模并行填写，停机降到 660 s。雕刻要素随之从 1056 s 降到 756 s，机加工从 1630.2 s 降到 1330.2 s。

带内外部标注的要素表导出 Excel，作为 SMED 分析和《头管雕刻换模标准作业指导书》的底稿。人工逐帧切分一段 16 min 的换模录像要一个多小时，工程师现在把这段时间用在内外部判断和 ECRS 讨论上。

头管雕刻换模的 SMED 三步（基于 AI 常规分析的 35 个要素）

现状 960 s 全部在停机中完成；内外分离后停机 744 s，再经 ECRS 优化降到 660 s



【图 47 头管雕刻换模的 SMED 三步】

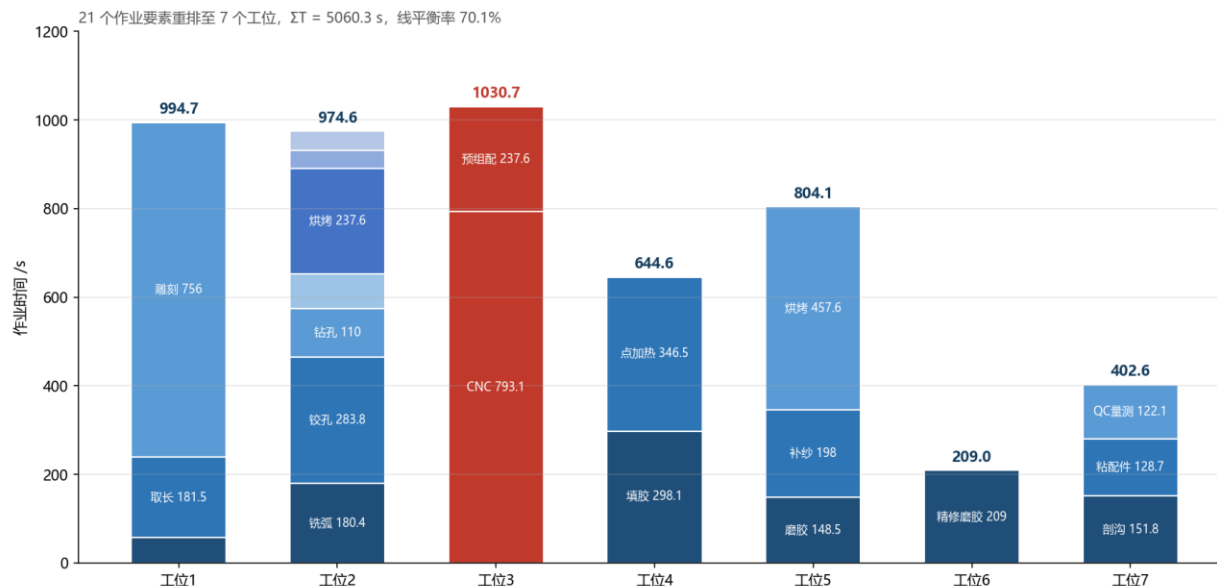
4. 工位重排：软件外建模求解，山积图可视化

工位重排用 0-1 整数规划，建立 147 个变量的模型，在满足作业先后顺序的前提下以瓶颈工位时间最小为目标，用 Lingo 求解，这一步在知绘 IE 之外完成。小组把求得的工位划分交给智能体，用山积图模板画成可视化的山积图，21 个要素按新工位堆叠，瓶颈 1030.7 s、线平衡率约 70.1%，各工位负荷的高低一眼可见，可直接用于汇报。划分如下表。

【表 16 0-1 整数规划的工位划分（SMED 改善后）】

工位	包含的作业要素	工位时间 /s
1	水喷砂、取长、雕刻	994.7
2	铣弧、铰孔、钻孔、超声、烘烤、削毛边、QC	974.6
3	CNC、预组配	1030.7
4	填胶、点加热	644.6
5	磨胶、补纱、烘烤	804.1
6	精修磨胶	209.0
7	剖沟、粘配件、QC 量测	402.6
合计	21 个作业要素	5060.3

X复材车架线改善后山积图：工位3 1030.7 s 为瓶颈



【图 48 车架线改善后山积图】

瓶颈 1030.7 s 仍高于 480 s 的订单节拍，需要靠多人并行维持。方案的动态效果要靠仿真验证。知绘 IE 不做仿真，小组把山积图对应的要素工时表导出 Excel，直接作为 FlexSim 模型的输入：改善后单班产出从 17 件升到 24 件，增长 41.2%，设备平均利用率从 53.7% 升到 67.0%。过去仿真输入要从各工序的测时表里手工抄录要素工时，约半小时；现在要素表导出即可使用。

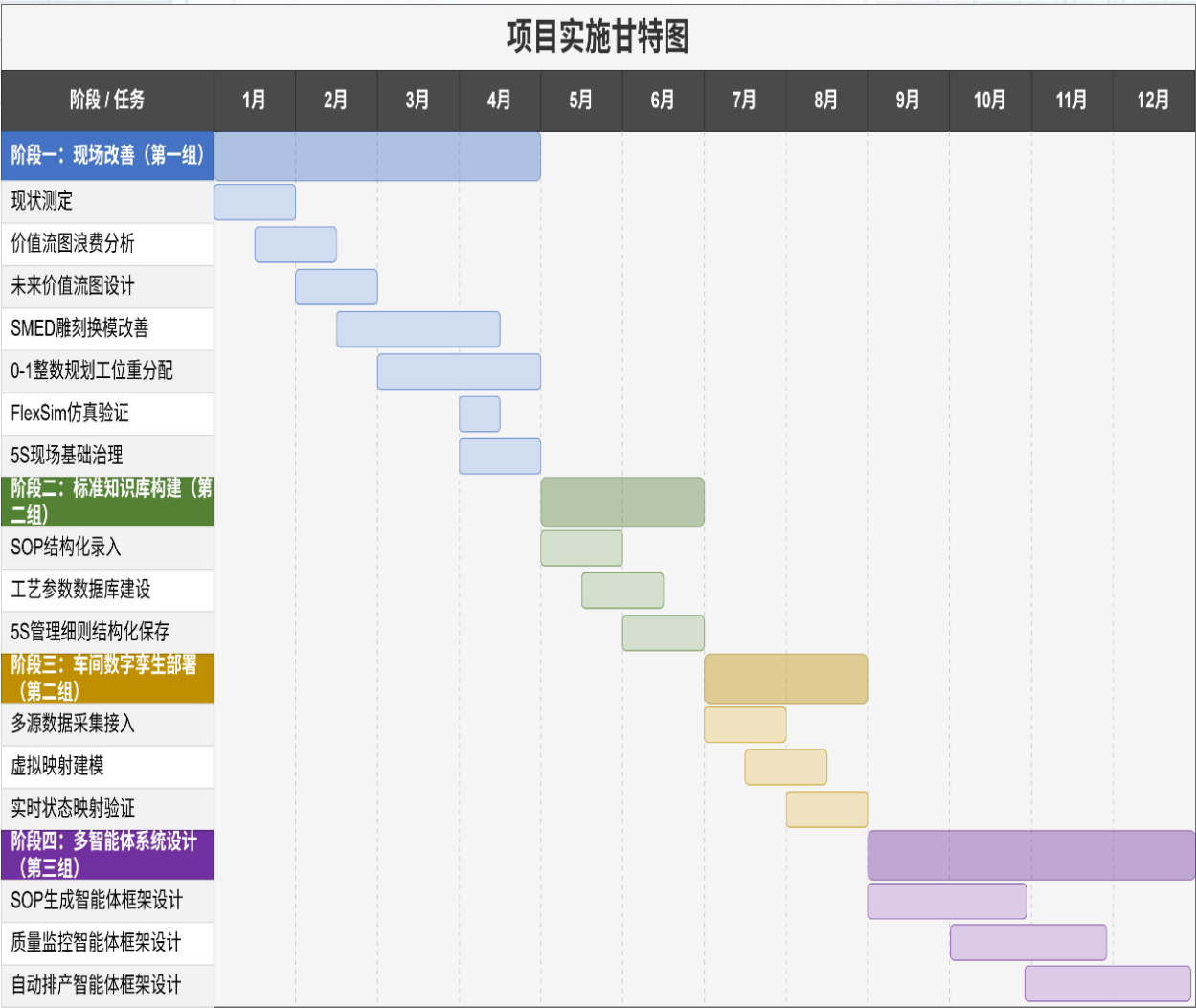
5. 标准化与知识沉淀

改善后的要素表作为新的标准时间基线，小组用产线 SOP 工具重新生成各工序的作业指导书，作业照片从录像截帧。5S 推行中，小组为工具设置定置托盘和挂架，固定物料存放位置，划分车架停放区，编写《5S 现场管理细则》，规定每月由车间主任牵头复盘，按红黄绿三色标记偏差项，同时培养覆盖 2~3 道工序的多能工。



【图 49 5S 整顿后的工具定置与车架停放区】

《头管雕刻换模标准作业指导书》和《5S 现场管理细则》上传到知识库，经隐私扫描、抽取、分页和去重后入库，操作员和新员工可以直接在对话中问“换模第几步做什么”，回答附细则原文出处。项目整体排期也用甘特图模板生成，全程 12 个月，如下图。



【图 50 精益改善与数字化升级项目甘特图（绘图模板：甘特图）】

6. 知绘 IE 替代了哪些重复工作

【表 17 IE 团队的重复工作与知绘 IE 的替代】

重复工作	传统做法（估算）	知绘 IE 的做法	仍由工程师完成
画价值流图、山积图、鱼骨图	Visio 或 Excel 手绘，每张半天到 1 h	数据或口述交给智能体，模板出图，可在 draw.io 修改	确定画什么，判断数据是否可信
拆换模录像	逐帧切分 16 min 录像，1 h 以上	AI 初稿 35 个要素，复核即可	内外部作业判断、ECRS
整理仿真输入	手工抄录要素工时	要素工时表导出 Excel	FlexSim 建模与结果解读
编作业指导书	手工排版、现场拍照	按文件夹生成带截帧的 SOP	审核签字
查阅制度文件	翻纸质细则	细则入知识库，问答附出处	制定与修订细则

这个案例里，方法选择、建模和决策都由工程师完成，知绘 IE 接下了其中可以按规则完成的部分：出图、拆录像、导表和归档。改善方法本身没有变，团队的时间从画图和整理表格转到了判断和验证上。

九、结语

（一）技术创新与可扩展性

知绘 IE 的创新不在某一个算法，而在大模型和 IE 方法的分工方式。我们坚持“AI+ 而非 +AI”：模型负责理解任务、追问缺失参数、编排工具和组织文字，所有数字由模块后端的确定性算法算出，模型只引用不复算。对话里、页面上和导出文件里的数字因此始终一致，工程师可以拿着公式逐项复核。围绕这条分工线，每一项产出都可以回查和修改：回答里的引用能点开回到原段落，视频分析的结果是可以逐个编辑的要素版本，AI 结果和人工修改留在同一条版本链上。

可扩展性是知绘 IE 在架构上最看重的一点。我们把智能体理解为“模型加支撑体系”：模型负责推理，支撑体系负责其余的一切，包括工具、记忆、安全边界、任务编排和状态管理。模型可以随时替换，支持 OpenAI 兼容接口、DeepSeek、Anthropic 和本机 Ollama 部署的开源模型；支撑体系中的每一部分也都是可替换、可增补的部件。知识库、绘图、工时分析和生图都以插件形式挂在底座上，新增一个业务模块只需提供一组工具和一份使用说明，不需要改动核心；技能把可复用的做事方法封装成说明文件，可以在对话中按需安装；MCP 服务把联网检索等外部能力接进来；绘图模块的模板以模板包形式加载，新增一类专业图只需放入一个模板包。



【图 51 知绘 IE 的扩展结构】

这套结构为知绘 IE 走向工业工程领域的生态打下了基础。今后可以由企业、高校和顾问贡献各自的插件、技能、知识书和模板包：企业把本厂的计量口径和报告格式沉淀为项目规约和技能，高校把教材编译成知识书，顾问把诊断方法做成可复用的技能，设备和系统厂商以插件或 MCP 服务接入现场数据。所有扩展都运行在同一套安全边界和引用规范之内，数字仍由确定性算法给出，生态越大，工程师手里可用的方法和工具越多，而结果的可复核性不会因此打折扣。

技术实现上，知绘 IE 本地优先，安装包自带约 1.09 GB 的离线运行时，所有服务只监听本机地址，适合网络条件一般、IT 支持薄弱的工厂。各子项目共有 1,918 项自动化测试，通过率 98.9%，核心计算用例全部覆盖。

（二）方案价值与社会价值

知绘 IE 的用户首先是中小制造企业里兼做 IE 的生产主管、工艺员和班组长，其次是有专职 IE 但人手紧张的中型工厂。他们工位多、改善需求零散、没有时间逐帧测时。知绘 IE 用对话代替菜单，让一次工位改善从“找专家、买软件、手工整理”变成“录像、放进文件夹、复核初稿”。

我国制造业中小企业数量多，精益改善的方法早已成熟，难的是没有人、没有工具去做。知绘 IE 降低的是做一次改善的门槛，系统给出初稿，工程师把精力放在判断上。很多工厂正是因为担心录像和工艺资料外流才不敢用云端工具，知绘 IE 配置本机开源模型时可以做到数据完全不出厂，让对数据敏感的企业也能用上大模型。在 IE 教学中，学生可以拿真实录像练习，再把自己的标注和 AI 初稿对照，比只看教材例题更接近现场。课程实验案例中，学生小组用知绘 IE 统一口径后，发现手工表格的线平衡率被高估了 13.18 个百分点，这类口径问题在纯手工汇总中很难发现。

（三）局限与下一步

第一，目前插件较少。现有插件覆盖知识检索、专业绘图、工时分析和生图四类能力，排产仿真、设施布局、人因评价、质量数据采集等 IE 常见场景还没有对应插件。下一步将按 IE 改善链路逐步补齐，并开放插件的开发规范，让更多参与者能够贡献插件。

第二，插件需求有待优化。现有插件的工具划分和使用说明主要依据开发团队对 IE 工作流的理解设计，还需要在真实使用中持续打磨，例如自动分析在提交时补充工序和物料提示、保存时对默认评比系数和宽放率给出复核提醒、长视频分析支持断点续跑等。我们将收集工程师的使用反馈，按实际需求调整工具粒度和交互方式。

第三，应用案例仍需扩充。目前的三个案例覆盖离散装配教学、竹制品加工和碳纤维复材加工，都是单条产线的数据，部分改善效果来自计算和仿真，尚未经过长期现场复测，课程实验的场景也比工厂简单。还需要在缝制、电子装配等更多行业的工厂中试用，并跟踪更长时间的改善保持情况，用更多真实场景检验和完善这套设计。

附录 A 运行环境与第三方组件

知绘 IE 是 Windows 桌面应用，安装包自带离线运行时，下表列出运行所需的环境条件。

【表 18 运行环境要求】

项目	要求
操作系统	Windows 10 或 Windows 11, x64
内存	8 GB 以上
显卡	不需要独立显卡，模型通过接口调用；改用本机 Ollama 部署模型时，按所选模型的要求配置
磁盘	运行时解包后约 1.09 GB，另需数据目录空间，作业视频按原文件大小保存
网络	安装和本机功能不需要联网；调用模型接口、联网检索时需要访问用户配置的地址
前置软件	无需另装 Python、数据库或浏览器，所需组件均随安装包提供
模型	对话、向量、视觉、生图四类模型分别配置，支持 OpenAI 兼容接口、DeepSeek、Anthropic 和本机 Ollama

知绘 IE 随附的主要第三方组件如下，均为宽松型开源许可，允许修改和再分发，要求保留版权声明。安装包不附带任何模型权重，模型的使用条款由用户与模型服务方约定。

【表 19 主要第三方组件与许可】

组件	版本	用途	许可
Electron	37.10.3	桌面程序、进程管理	MIT
开源智能体框架	0.84.4	对话循环、工具调度、会话管理、上下文压缩	MIT
Next.js	16.3.1	智能体工作台	MIT
Vue	3.5	三个业务模块的页面	MIT
Element Plus	2.8 / 2.13	界面组件	MIT
ECharts	6	知识图谱与看板图表	Apache-2.0
draw.io	32.0.2	流程图引擎（离线内嵌）	Apache-2.0
Python 嵌入版	3.12.10	业务后端与办公脚本的解释器	PSF

组件	版本	用途	许可
FastAPI	0.136.0 / 0.141.1	业务模块接口服务	MIT
ChromaDB	0.5.15	知识库向量索引	Apache-2.0
matplotlib	3.10.8	绘图模板出图	PSF 类
imageio-ffmpeg	0.6.0	作业视频抽帧	BSD-2-Clause
openpyxl	3.1.5	Excel 导出	MIT
python-docx	1.2.0	Word 文档生成	MIT
pandas	2.3.3	数据处理与统计分析	BSD-3-Clause

附录 B 核心代码节选

本附录节选知绘 IE 中直接决定产出数字和数据可靠性的 32 段核心代码，覆盖计算口径、检索、判重、自动分析、桌面外壳、智能体集成、知识库入库和绘图同步等环节，均为实际运行代码的原文，省略部分以 “...” 表示。

B.1 标准时间计算与银行家舍入

要素的观测时间先扣除被打断的无效区间，再乘评比系数和宽放率，浪费类要素计为 0。舍入采用与后端一致的银行家舍入，保证页面、对话和导出文件中的毫秒数逐位相同。

```
function invalidMs(segment: AnalysisSegment): number {
  if (segment.invalid_start_ms === null || segment.invalid_end_ms === null) return 0
  const start = Math.max(segment.start_ms, segment.invalid_start_ms)
  const end = Math.min(segment.end_ms, segment.invalid_end_ms)
  return Math.max(0, end - start)
}

function roundLikePython(value: number): number {
  const lower = Math.floor(value)
  const fraction = value - lower
  if (fraction === 0.5) return lower % 2 === 0 ? lower : lower + 1
  return Math.round(value)
}

export function segmentTimes(segment: AnalysisSegment): SegmentTimes {
  const measuredMs = Math.max(0, segment.end_ms - segment.start_ms)
  const wastedMs = invalidMs(segment)
  const validMs = Math.max(0, measuredMs - wastedMs)
  const standardMs = segment.category === 'waste'
    ? 0
    : roundLikePython(validMs * segment.rating * (1 + segment.allowance))
  return { measuredMs, invalidMs: wastedMs, validMs, standardMs }
}
```

B.2 线平衡率计算

瓶颈取工序时间的最大值，线平衡率等于工序时间总和除以瓶颈时间与工序数之积。工程师在山积图上每拖动一次要素，都会调用这段代码重新计算。

```
export function calculateBalance(processes: BalanceProcess[]): BalanceMetrics {
  const totals = processes.map(processTotalMs)
  const processCount = processes.length
  const elementCount = processes.reduce(
    (total, process) => total + process.elements.length, 0)
  const totalMs = totals.reduce((sum, value) => sum + value, 0)
  const bottleneckMs = totals.length ? Math.max(...totals) : 0
  const bottleneckIndex = totals.findIndex((value) => value === bottleneckMs)
```

```

const bottleneck = bottleneckIndex >= 0 ? processes[bottleneckIndex] : undefined
return {
  processCount,
  elementCount,
  totalMs,
  bottleneckMs,
  bottleneckName: bottleneck ? `工序${bottleneck.index} ${bottleneck.name}` : '-',
  averageMs: processCount ? totalMs / processCount : 0,
  balanceRate: bottleneckMs > 0 ? (totalMs / (bottleneckMs * processCount)) * 100 : 0,
  totals,
}
}
}

```

B.3 混合检索的融合与重排

关键词和语义两路结果先按名次做倒数排名融合，再按关键词得分、语义相似度、标题章节重合度和命中次数四项加权重排，同一页面最多保留 3 段。整个过程不调用大模型，同一问题的结果固定可重放。

```

for rank, item in enumerate(lexical, start=1):
    row = merged.setdefault(_norm_chunk_id(item), dict(item))
    row["lexical_rank"] = rank
    row["lexical_score"] = float(item.get("lexical_score") or 0)
    row["fusion_score"] = row.get("fusion_score", 0.0) + _rrf(rank)
for rank, item in enumerate(semantic, start=1):
    row = merged.setdefault(_norm_chunk_id(item), dict(item))
    row["semantic_rank"] = rank
    row["semantic_score"] = float(item.get("semantic_score") or 0)
    row["fusion_score"] = row.get("fusion_score", 0.0) + _rrf(rank)
...
row["rerank_score"] = round(
    0.35 * lexical_norm
    + 0.35 * semantic_score
    + 0.2 * overlap
    + 0.1 * min(1.0, int(row.get("hit_count") or 0) / 3),
    6,
)
...
ranked = sorted(
    merged.values(),
    key=lambda x: (-x["rerank_score"], -x.get("fusion_score", 0), x["chunk_id"]),
)
out: list[dict] = []
per_page: dict[str, int] = {}
for row in ranked:
    slug = str(row.get("slug") or "")
    if per_page.get(slug, 0) >= per_page_cap:
        continue
    per_page[slug] = per_page.get(slug, 0) + 1
    out.append(row)
    if len(out) >= top_k:
        break
return out

```

B.4 实体三级判重

新实体依次经过名称、别名和向量余弦三关。余弦相似度不低于 0.92 时自动合并，0.75 至 0.92 之间新建页面并记录合并建议，交由维护者确认。

```
# 1) 名称命中
if candidate_name in existing_set:
    return candidate_name, "merged", {"matched_via": "exact_slug"}

# 2) 别名词表命中
alias_canonical = find_alias_match(candidate_name, alias_groups=alias_groups)
if alias_canonical and alias_canonical in existing_set:
    return alias_canonical, "merged_via_alias", {
        "matched_via": "alias",
        "alias_group_canonical": alias_canonical,
    }
if alias_canonical:
    return alias_canonical, "created", {
        "matched_via": "alias_new",
        "alias_group_canonical": alias_canonical,
    }

# 3) 向量余弦比对
if not existing or not candidate_brief.strip():
    return candidate_name, "created", {"matched_via": "no_existing_or_brief"}
...
best_slug, best_score = max(scores, key=lambda x: x[1]) if scores else (None, 0.0)
meta = {"cosine": round(best_score, 4), "matched_against": best_slug}

if best_slug and best_score >= auto_merge_threshold:
    return best_slug, "merged_via_similarity", meta
if best_slug and best_score >= suggest_threshold:
    meta["note"] = "suggested_merge_recorded"
    return candidate_name, "suggested_merge", meta
return candidate_name, "created", meta
```

B.5 自动分析的短段归并

视觉模型逐帧识别出的区间往往很碎。这段规则后处理把相邻的相同动作合成一段，把短于 200 毫秒的碎片并入相邻要素，得到可以直接在时间轴上编辑的要素初稿。

```
def _merge(spans: list[tuple[int, int, Any]], signature) -> list[tuple[int, int, Any]]:
    """过短的区间并入前一段（首段并入后一段），连续相同内容合成一段。"""
    merged: list[list[Any]] = []
    for start, end, row in spans:
        if merged and signature(merged[-1][2]) == signature(row):
            merged[-1][1] = end
            continue
```

```

merged.append([start, end, row])
result: list[list[Any]] = []
for item in merged:
    if result and item[1] - item[0] < MIN_SEGMENT_MS:
        result[-1][1] = item[1]
    else:
        result.append(item)
if len(result) > 1 and result[0][1] - result[0][0] < MIN_SEGMENT_MS:
    result[1][0] = result[0][0]
    result.pop(0)
final: list[list[Any]] = []
for item in result:
    if final and signature(final[-1][2]) == signature(item[2]):
        final[-1][1] = item[1]
    else:
        final.append(item)
return [(start, end, row) for start, end, row in final]

```

B.6 分析版本快照与冲突检查

每次保存追加一个版本号加一的完整快照。提交所依据的版本已不是最新时返回冲突；两个请求同时写入、撞上数据库的唯一约束时，先回滚再同样返回冲突，保证任何一方的修改都不会被无声覆盖。

```

async def _write_revision(
    analysis_id: str,
    payload: AnalysisPayload,
    session: AsyncSession,
) -> dict[str, object]:
    if payload.revision is None:
        raise HTTPException(status_code=status.HTTP_422_UNPROCESSABLE_CONTENT,
            detail="revision is required")
    current = await _current_analysis(session, analysis_id)
    if current is None:
        raise HTTPException(status_code=status.HTTP_404_NOT_FOUND, detail="analysis was not found")
    if payload.revision != current.revision:
        raise HTTPException(status_code=status.HTTP_409_CONFLICT, detail="analysis revision is stale")

    duration_ms = await _ensure_video_exists(session, payload.video_asset_id)
    try:
        snapshot = _snapshot_from_payload(payload, current.analysis_key,
            revision=current.revision + 1)
        normalize_segments(snapshot.mode, snapshot.tracks, duration_ms)
    except TimelineValidationError as error:
        raise HTTPException(status_code=status.HTTP_422_UNPROCESSABLE_CONTENT,
            detail=str(error)) from error
    session.add(snapshot)
    try:
        await session.commit()
    except IntegrityError as error:
        await session.rollback()

```

```
    raise HTTPException(status_code=status.HTTP_409_CONFLICT, detail="analysis revision
is stale") from error
    return {"data": _analysis_data(snapshot), "error": None}
```

B.7 后端进程拉起与健康检查

桌面壳为每个工具模块先探测端口是否空闲，再用便携 Python 运行时以 uvicorn 拉起后端，并把配置里的 `__DATA__` 占位替换为用户数据目录。启动后按秒轮询健康接口，超时的模块记入异常列表，界面据此提示用户而不是给出空白页。（shell/main.js）

```
// ----- 端口 -----
function probePort(port) {
  return new Promise((resolve) => {
    const srv = net.createServer();
    srv.once('error', () => resolve(false));
    srv.once('listening', () => srv.close(() => resolve(true)));
    srv.listen(port, '127.0.0.1');
  });
}
...
// ----- 后端进程 -----
async function startBackend(mod) {
  const b = mod.backend;
  // 打包版用便携 Python 运行时(runtime/), 开发版退回模块内 venv
  const venvPy = b.python ? resolveRuntimePath(b.python) : path.join(ROOT, b.venv, 'Scripts',
'python.exe');
  const port = await requireAvailablePort(b.port, probePort, `${mod.id} backend`);
  const dataRoot = await provision(mod);
  const env = { ...process.env, ...(b.env || {}), ...AI_CONFIG_ENV };
  for (const [k, v] of Object.entries(env)) {
    if (typeof v === 'string') {
      env[k] = v.replaceAll('__DATA__', dataRoot).replaceAll('__RESOURCES__', ROOT);
    }
  }
  const pythonDir = path.dirname(venvPy);
  const pythonPath = [pythonDir];
  if (b.sitePackages) pythonPath.push(resolveRuntimePath(b.sitePackages));
  if (env.PYTHONPATH) pythonPath.push(env.PYTHONPATH);
  env.PYTHONPATH = pythonPath.join(path.delimiter);
  log(`[${mod.id}] startBackend: spawn ${venvPy} :${port}`);
  const child = spawn(venvPy, ['-m', 'uvicorn', b.app, '--host', b.host, '--port', String(port)], {
    cwd: path.join(ROOT, b.cwd), env, stdio: ['ignore', 'pipe', 'pipe'], windowsHide: true,
  });
  state.backends.set(mod.id, child);
  child.stdout.on('data', (d) => log(`[${mod.id}:out] ${d.toString().trim()}`));
  child.stderr.on('data', (d) => log(`[${mod.id}:err] ${d.toString().trim()}`));
  child.on('exit', (code) => log(`[${mod.id}] 后端进程退出 code=${code}`));

  const healthUrl = `http://127.0.0.1:${port}${b.health || '/health'}`;
  for (let i = 0; i < 90; i++) {
    await new Promise((r) => setTimeout(r, 1000));
    if (child.exitCode !== null) break;
    try {
      const res = await fetch(healthUrl);
```

```

    if (res.status < 500) { log(`[${mod.id}] 后端就绪 ${healthUrl} (${res.status})`); return
port; }
    } catch {}
  }
  // 起不来时以前只写日志就继续，界面照常打开、每个请求 502，用户只看到空白页无从判断。
  state.unhealthy.push(mod.id);
  log('[ ' + mod.id + ' ] 后端健康检查超时，请查看日志 ' + LOG_FILE);
  return port;
}

```

B.8 种子数据逐文件填充

首次安装时把自带知识库等种子内容逐文件拷入用户数据目录，单个文件失败重试三次，全部成功才写 .seeded 标记。这样杀毒软件拦截个别文件时，其余文件照常落盘，下次启动只补缺的部分。
(shell/main.js)

```

// ----- 数据目录外置 + 种子数据 -----
// 只在首装填充一次自带内容，填写写 .seeded 标记，之后完全交给用户：改了、删了都不再插手。
// 判据用标记而不是「目标目录是否存在」—后端启动时自己会建出空的 wiki 子目录，
// 拿目录存在当判据会让自带内容永远进不来（老版本就是这么漏的）。
// 逐文件拷，不用 cpSync 整棵树一把梭：杀软/占用只拦住一个文件时，
// cpSync 会中止全部 1650 个文件且只报一个顶层路径；逐文件拷则只丢那一个，
// 其余照常落盘，下次启动自动跳过已有的、只补剩下的，进度能累积。
async function copyTreeResilient(mod, from, to, failures) {
  let copied = 0;
  let skipped = 0;
  const stack = [['', fs.statSync(from).isDirectory()]];
  while (stack.length) {
    const [rel, isDir] = stack.pop();
    const src = path.join(from, rel);
    const dst = path.join(to, rel);
    if (isDir) {
      try {
        fs.mkdirSync(dst, { recursive: true });
      } catch (e) {
        if (e.code !== 'EEXIST') {
          failures.push(`${rel} || '.' (mkdir ${e.code} || '') ${e.message}`);
          continue;
        }
      }
    }
    let entries;
    try {
      entries = fs.readdirSync(src, { withFileTypes: true });
    } catch (e) {
      failures.push(`${rel} || '.' (readdir ${e.code} || '') ${e.message}`);
      continue;
    }
    for (const ent of entries) stack.push([path.join(rel, ent.name), ent.isDirectory()]);
    continue;
  }
  if (fs.existsSync(dst)) { skipped++; continue; } // 用户已有的一律不动
  let ok = false;
  for (let attempt = 1; attempt <= 3 && !ok; attempt++) {

```

```

    try {
      fs.copyFileSync(src, dst);
      ok = true;
      copied++;
    } catch (e) {
      if (attempt === 3) {
        failures.push(`${rel} (${e.code} || '') errno=${e.errno ?? ''} ${e.syscall} || ''`
`${e.message}));`);
      } else {
        await new Promise((r) => setTimeout(r, attempt * 300));
      }
    }
  }
}
log(`${mod.id}] 填充 ${to}: 新增 ${copied}, 跳过已有 ${skipped}, 失败 ${failures.length}`);
return { copied, skipped };
}

async function provision(mod) {
  const dataRoot = path.join(USER_DATA, mod.dataDir || mod.id);
  fs.mkdirSync(dataRoot, { recursive: true });
  const seeds = mod.seed || (mod.backend && mod.backend.seed) || [];
  if (!seeds.length) return dataRoot;
  const stamp = path.join(dataRoot, '.seeded');
  if (fs.existsSync(stamp)) return dataRoot;
  const failures = [];
  for (const s of seeds) {
    const from = path.join(ROOT, s.from);
    if (!fs.existsSync(from)) continue;
    await copyTreeResilient(mod, from, path.join(dataRoot, s.to), failures);
  }
  // 只有一个文件都没失败才落标记。否则下次启动继续补剩下的——
  // 一次瞬时拦截不该让这台机器永久空库。
  if (!failures.length) {
    fs.writeFileSync(stamp, new Date().toISOString(), 'utf8');
    return dataRoot;
  }
  state.seedFailures.push({ id: mod.id, count: failures.length });
  log(`${mod.id}] 有 ${failures.length} 个文件没拷成功, 不写 .seeded, 下次启动会补。前 20 个: `);
  for (const f of failures.slice(0, 20)) log(`${mod.id}]   ${f}`);
  return dataRoot;
}
}

```

B.9 公开工具清单与系统代理

生成给前端的工具清单时只放名称、图标和跳转地址，不带任何登录令牌，令牌由 `/__desktop_enter__` 入口单独注入。另把 Electron 解析出的 PAC 代理转成子进程可读的环境变量，让联网检索在公司代理下可用。（`shell/platform.js`）

```

function resolveDesktopEntrySession(mod, authTokens) {
  const authId = mod.authRef || (mod.auth ? mod.id : null);
  if (!authId) return { ready: true, session: null };
  const session = authTokens.get(authId) || null;

```

```

    return { ready: session !== null, session };
  }

  async function requireAvailablePort(port, probe, label = 'Configured') {
    if (!await probe(port)) {
      throw new Error(`${label} port ${port} is already in use`);
    }
    return port;
  }

  function createPublicManifest(config, ports) {
    const list = config.modules || [];
    const homeModule = (list.find((module) => module.home) || list[0])?.id;
    const homePort = ports.get(homeModule)?.frontend;
    const tools = (config.tools || []).map((tool) => {
      const port = ports.get(tool.module)?.frontend;
      const path = sanitizeRedirect(tool.path || '/');
      const href = tool.module === homeModule
        ? path
        : `http://127.0.0.1:${port}/__desktop_enter__?redirect=${encodeURIComponent(path)}`;
      return {
        id: tool.id,
        name: tool.name,
        description: tool.description || '',
        icon: tool.icon || '',
        href,
        module: tool.module,
      };
    });
    return {
      appName: config.appName || '知绘 IE',
      portalHref: 'http://127.0.0.1:' + homePort + '/',
      tools,
    };
  }

  // Node 的 fetch 不读 Windows 系统代理: 把 Electron resolveProxy 的 PAC 结果转成子进程环境变量
  function proxyEnvFromPac(pac, env = {}) {
    if (env.HTTPS_PROXY || env.https_proxy || env.HTTP_PROXY || env.http_proxy) return {};
    const first = String(pac || '').split(';')[0].trim();
    const m = /^(PROXY|HTTPS)\s+([\w.-]+:\d+)$/i.exec(first);
    if (!m) return {};
    const proxy = `${m[1].toUpperCase() === 'HTTPS' ? 'https' : 'http'}://${m[2]}`;
    const noProxy = [...new Set(['127.0.0.1', 'localhost', '::1', ...String(env.NO_PROXY || env.no_proxy || '').split(',')])
      .map((s) => s.trim()).filter(Boolean)].join(',');
    return { HTTPS_PROXY: proxy, HTTP_PROXY: proxy, NO_PROXY: noProxy, NODE_USE_ENV_PROXY: '1' };
  }

```



```

const raw = String(command ?? "").toLowerCase();
for (const [re, label, devOnly] of DANGEROUS) {
  if (full && devOnly) continue;
  if (re.test(raw) || re.test(lower)) return label;
}
if (/\\brm\\s+-[a-z]*r[a-z]*f?[a-z]*\\s+(\\|~|\\|\\*|\\$home)\\s*(\\|;|&)/.test(raw)) return "删除整个磁盘或用户目录";
for (const stmt of statements(lower)) {
  if (DELETE_VERB.test(stmt) && ROOT_TARGETS.some((re) => re.test(stmt))) return "删除整个磁盘或用户目录";
  if (MUTATE_VERB.test(stmt) && SYSTEM_TARGETS.test(stmt)) return "修改系统目录";
}
return "";
}

```

B.11 工具调用守卫

智能体每次调用读写或命令工具前先经过这个纯函数：软件数据目录读写都禁止，安装目录只读，系统目录不可改。技能目录内的脚本路径先剔除再检查，带 .. 的路径保留，防止借技能目录跳到密钥文件。
(modules/agent/web/lib/zhahui/guard.mjs)

```

const READ_TOOLS = new Set(["read", "ls", "grep", "find"]);

// 技能自带脚本要能执行（python <技能目录>\scripts\x.py），先把指向技能目录内的路径去掉再查数据目录；
// 带 .. 的路径保留，防止借技能目录跳到 auth.json 等文件
function withoutSkillPaths(lower, readablePaths) {
  let out = lower;
  for (const dir of readablePaths) {
    const r = path.resolve(dir).toLowerCase().replace(/[\.*?^${}()|[\]\|\|]/g, "\\$&");
    out = out.replace(new RegExp(`${r}{\\\\\\\\[^\\\\s";|&<>]*)?`, "g"), (m) => (m.includes("..") ? m : " "));
  }
  return out;
}

export function evaluateToolCall(event, policy) {
  const { toolName, input = {} } = event;
  const { cwd, protectedPaths = [], readablePaths = [], mode = "auto" } = policy;
  const full = mode === "full";
  // 完全权限放开安装目录只读；软件数据目录和系统目录始终保护
  const readonlyPaths = full ? [] : policy.readonlyPaths || [];
  const inProtected = (p) => protectedPaths.some((root) => isInside(p, root));
  const inReadonly = (p) => readonlyPaths.some((root) => isInside(p, root));

  if (READ_TOOLS.has(toolName)) {
    const target = resolveToolPath(cwd, input.path);
    if (inProtected(target) && !readablePaths.some((root) => isInside(target, root))) return
    { action: "block", reason: `不允许访问软件数据目录: ${target}` };
    return { action: "allow" };
  }

  if (toolName === "write" || toolName === "edit") {
    const target = resolveToolPath(cwd, input.path);

```

```

    if (inProtected(target)) return { action: "block", reason: `不允许修改软件数据目录: ${target}` };
    if (inReadonly(target)) return { action: "block", reason: `安装目录只读: ${target}` };
    if (systemPaths().some((root) => isInside(target, root))) return { action: "block", reason: `不允许修改系统目录: ${target}` };
    return { action: "allow" };
  }

  if (toolName === "powershell" || toolName === "bash") {
    const command = String(input.command ?? "");
    const lower = withoutSkillPaths(command.toLowerCase().replace(/\\/g, "\\\"), readablePaths);
    const roots = protectedPaths.filter((p) => !protectedPaths.some((q) => q !== p && isInside(p, q)));
    for (const root of roots) {
      const r = path.resolve(root).toLowerCase();
      const name = path.basename(r);
      if (lower.includes(r) || (name.length >= 8 && lower.includes(name))) {
        return { action: "block", reason: "命令涉及软件数据目录, 已拦截" };
      }
    }
    const danger = dangerousCommand(command, { full });
    if (danger) return { action: "block", reason: `危险命令已拦截 (${danger})。换一种不破坏系统/数据的做法, 或请用户自己执行。` };
    return { action: "allow" };
  }

  // 其余是 pi 自带或用户装的扩展工具 (subagent 等), 由各自的实现负责
  return { action: "allow" };
}

```

B.12 智能体扩展注册

把知绘 IE 作为 pi 智能体的扩展注册: 登记模型提供方、补齐温度等采样参数、挂载各工具插件, 并在工具调用前做守卫与询问模式确认, 调用后把生成的文件做成卡片, 新写的 .drawio 自动存入知绘图。
(modules/agent/web/lib/zhihui/extension.mjs)

```

export function createZhihuiExtension() {
  return {
    name: "zhihui-ie",
    hidden: true,
    factory: (pi) => {
      const { cfg, services, plugins } = getZhihui();
      const provider = zhihuiProviderConfig(cfg);
      if (provider) pi.registerProvider(ZHIHUI_PROVIDER_ID, provider.config);

      // pi 只在调用方显式传 temperature 时才发, AI 设置里的温度和额外参数要在这里补进请求体
      pi.on("before_provider_request", (event, ctx) => {
        if (!provider || ctx?.model?.provider !== ZHIHUI_PROVIDER_ID) return undefined;
        return withSamplingParams(event.payload, provider.llm);
      });

      for (const tool of plugins.tools) {
        pi.registerTool({
          ...tool.definition,

```

```

        execute: (toolCallId, params, signal, onUpdate, ctx) => tool.run(params, {
            signal, onUpdate, toolCallId, cfg,
            sessionId: ctx?.sessionManager?.getSessionId?().(),
            sessionManager: ctx?.sessionManager,
        })),
    });
}

// 不走插件的 toResult: 要把 terminate 原样交给 pi, 提问后本轮结束等用户回答
pi.registerTool({ ...askUserTool, execute: async (_id, params) => runAskUser(params) });
pi.registerTool({
    ...skillInstallTool,
    execute: async (_id, params, _signal, _onUpdate, ctx) => runSkillInstall(params, {
        agentDir: getAgentDir(),
        cwd: ctx?.cwd || cfg.workspace,
    }),
});

const fileChanges = createFileChangeTracker({ exportDir: cfg.exportDir });

pi.on("tool_call", async (event, ctx) => {
    const cwd = ctx.cwd || cfg.workspace;
    const mode = readPermissionMode(ctx.sessionManager);
    const verdict = evaluateToolCall(event, {
        cwd,
        mode,
        protectedPaths: cfg.protectedPaths,
        readonlyPaths: cfg.readonlyPaths || [],
        readablePaths: readableSkillDirs(pi, cfg),
    });
    if (verdict.action === "block") return { block: true, reason: verdict.reason };
    if (mode === "ask" && needsConfirmation(event.toolName)) {
        // 没有界面（子代理、后台任务）时没人能确认，按拒绝处理
        const { title, message } = describeToolCall(event);
        const ok = ctx.hasUI ? await ctx.ui.confirm(title, message, { signal: ctx.signal }) :
false;
        if (!ok) return { block: true, reason: "用户未允许执行这一步。停下来问用户想怎么做，不要换写法重试。" };
    }
    fileChanges.before(event, cwd);
    return undefined;
});

// 对话里按文件卡片展示本轮生成/修改的文件；新写的 .drawio 顺手存进知绘图
pi.on("tool_result", async (event, ctx) => {
    const files = fileChanges.after(event);
    if (!files) return undefined;
    const details = withFileChanges(event.details, files);
    const zt = services.has("zhihuitu") ? services.get("zhihuitu") : null;
    if (!zt || !files.some((f) => f.status !== "deleted" && isDrawioPath(f.path))) return
{ details };
    const { cards, notes } = await autoSyncDrawioFiles(zt, cfg, files, {
        sessionId: ctx?.sessionManager?.getSessionId?().(),
        signal: ctx?.signal,
    });
    if (cards.length) details.cards = [...(Array.isArray(details.cards) ? details.cards :

```

```

[[]], ...cards];
    return {
      details,
      ...(notes.length ? { content: [...(event.content || []), { type: "text", text: `\n[知绘图] ${notes.join("\n")}` } ] } : {}),
    };
  });
},
};
};
}

```

B.13 AI 配置加密与回退

所有工具共用一份 ai.json，API Key 用与 Python 端兼容的 Fernet 加密，密钥文件以 wx 方式创建防止两端各生成一把。各模型段未填地址或 Key 时回退到通用接口；保存时空串或屏蔽串表示不改，null 表示清除。（modules/agent/web/lib/zhihui/ai-config.mjs）

```

// 没有就生成；用 wx 防止和 Python 端同时生成两把不同的 key
function aiKey(cfg) {
  const file = aiKeyPath(cfg);
  try {
    const existing = fs.readFileSync(file, "utf8").trim();
    if (existing) return existing;
  } catch {}
  fs.mkdirSync(path.dirname(file), { recursive: true });
  try {
    fs.writeFileSync(file, newFernetKey(), { encoding: "utf8", flag: "wx" });
  } catch (e) {
    if (e?.code !== "EEXIST") throw e;
  }
  return fs.readFileSync(file, "utf8").trim();
}
...
function decryptKey(key, token) {
  if (!token) return "";
  try { return fernetDecrypt(key, token); } catch {
    // 手工写入的明文 key 照用；坏掉的密文当没配
    return String(token).startsWith("gAAAA") ? "" : String(token);
  }
}
...
/** 解密并套上 base 回退后的生效配置；未设置的字符串字段为空串。 */
export function resolveAiConfig(cfg) {
  const { sec, key } = decryptedSections(cfg);
  const base = { baseUrl: str(sec("base").baseUrl), apiKey: decryptKey(key, sec("base").apiKey) };
  const endpoint = (name) => ({
    baseUrl: str(sec(name).baseUrl) || base.baseUrl,
    apiKey: decryptKey(key, sec(name).apiKey) || base.apiKey,
  });
  const chat = sec("chat");
  const embedding = sec("embedding");
  const image = sec("image");
  const vision = sec("vision");
}

```

```

const extraParams = chat.extraParams && typeof chat.extraParams === "object"
&& !Array.isArray(chat.extraParams) ? chat.extraParams : {};
return {
  chat: {
    ...endpoint("chat"),
    model: str(chat.model),
    provider: str(chat.provider),
    temperature: num(chat.temperature, NUMBER_DEFAULTS.chat.temperature),
    maxTokens: num(chat.maxTokens, NUMBER_DEFAULTS.chat.maxTokens),
    contextWindow: num(chat.contextWindow, NUMBER_DEFAULTS.chat.contextWindow),
    extraParams,
  },
  embedding: { ...endpoint("embedding"), model: str(embedding.model), dim: num(embedding.dim,
NUMBER_DEFAULTS.embedding.dim) },
  image: { ...endpoint("image"), model: str(image.model), size: str(image.size) },
  vision: { ...endpoint("vision"), model: str(vision.model), maxFrames: num(vision.maxFrames,
NUMBER_DEFAULTS.vision.maxFrames) },
  tavily: { apiKey: decryptKey(key, sec("tavily").apiKey) },
  firecrawl: { apiKey: decryptKey(key, sec("firecrawl").apiKey), baseUrl:
str(sec("firecrawl").baseUrl) },
};
}
...
/**
 * 局部更新：只改传入的段和字段。apiKey 为空串、屏蔽串或省略表示不变，null 表示清除。
 * 调用方先用 validateAiConfigInput 校验。返回写入后的屏蔽视图。
 */
export function saveAiConfig(input, cfg) {
  const file = aiConfigPath(cfg);
  migrateAiConfig(cfg);
  const raw = readAiConfigRaw(cfg) || {};
  const next = { ...raw };
  for (const [name, fields] of Object.entries(SECTIONS)) {
    const patch = input?.[name];
    if (!patch || typeof patch !== "object") continue;
    const section = { ...(raw[name] && typeof raw[name] === "object" ? raw[name] : {}) };
    for (const field of fields) {
      if (!(field in patch) || patch[field] === undefined) continue;
      const value = patch[field];
      if (field === "apiKey") {
        if (value === null) delete section.apiKey;
        else if (typeof value === "string" && value.trim() && !isMasked(value)) section.apiKey =
fernetEncrypt(aiKey(cfg), value.trim());
      } else if (typeof value === "string") {
        section[field] = value.trim();
      } else {
        section[field] = value;
      }
    }
    next[name] = section;
  }
  next.version = Math.max(AI_CONFIG_VERSION, Number(raw.version) || 0);
  writeJsonAtomic(file, next);
  return maskedAiConfig(cfg);
}

```

B.14 知识库证据编号分配

同一会话内的知识库证据按 E1、E2... 连续编号，同一段再次命中复用旧编号，便于回答中引用。模型上下文里已给过全文的段只发编号，并用内存表兜住同一轮并行检索尚未写入会话的结果。

(modules/agent/web/lib/zhihui/plugins/kgqa/evidence.mjs)

```
const TOOL_NAMES = new Set(["kb_search", "kb_read"]);
const MAX_SESSIONS = 200;
// 已返回但还没写进会话的结果（同一轮并行调用），超时就不再算数
const PENDING_TTL_MS = 10 * 60 * 1000;
const memory = new Map(); // sessionId → { max, refs: Map<key, ref>, pending: Map<toolCallId, {keys, at}> }
...
function sessionMemory(sessionId) {
  const id = String(sessionId || "");
  let state = memory.get(id);
  if (state) {
    memory.delete(id); // 重新插入，Map 顺序即最近使用顺序
  } else {
    state = { max: 0, refs: new Map(), pending: new Map() };
    if (memory.size >= MAX_SESSIONS) memory.delete(memory.keys().next().value);
  }
  memory.set(id, state);
  return state;
}

/**
 * 给本次结果分配编号，并判断哪些段的全文模型已经看过。全程同步，并行调用不会撞号。
 * @returns 与 items 等长的 [{ref, brief}]
 */
export function assignRefs({ sessionId, sessionManager, toolCallId }, items) {
  const mem = sessionMemory(sessionId);
  const scanned = scanSessionEvidence(sessionManager);
  for (const [k, ref] of scanned.refs) if (!mem.refs.has(k)) mem.refs.set(k, ref);
  mem.max = Math.max(mem.max, scanned.max);

  // 拿不到会话就无从判断上下文里有什么，一律发全文
  const known = sessionManager ? scanSentText(sessionManager) : null;
  const sent = known ? new Set(known.sent) : new Set();
  if (known) {
    // 按整条分支判断是否已落盘：落盘后若又被压缩掉，不能再靠 pending 当成「已给」
    const written = writtenToolCallIds(branchEntries(sessionManager));
    const now = Date.now();
    for (const [id, p] of mem.pending) {
      if (written.has(id) || now - p.at > PENDING_TTL_MS) mem.pending.delete(id);
      else for (const k of p.keys) sent.add(k);
    }
  }

  const fresh = new Set();
  const full = new Set(); // 本次真正发了完整正文的段；clipped 的不算
  const out = items.map((it) => {
    const key = evidenceKey(it.slug, it.chunk);
    let ref = scanned.refs.get(key) || mem.refs.get(key);
```

```

    if (!ref) {
      mem.max += 1;
      ref = `E${mem.max}`;
      mem.refs.set(key, ref);
    }
    // 同一次结果里重复出现的段也只发一次全文
    const brief = sent.has(key) || fresh.has(key);
    if (!brief) {
      fresh.add(key);
      if (!it.clipped) full.add(key);
    }
    return { ref, brief };
  });
  if (known && toolCallId && full.size) mem.pending.set(toolCallId, { keys: full, at:
Date.now() });
  return out;
}

```

B.15 知识库检索工具

`kb_search` 调用知识库后端的检索接口，把命中段落渲染成带编号的证据块交给模型，并按页面去重生成跳转卡片。语义索引不可用时降级为关键词检索，并给用户一条可操作的提示。
(modules/agent/web/lib/zhahui/plugins/kgqa/index.mjs)

```

// 语义检索降级原因 → 给用户的提示；no_semantic_candidates 是语义腿正常但没命中，不打扰用户
const DEGRADED_NOTICES = {
  marker_missing: "知识库还没有建语义索引，本次只按关键词匹配。到 AI 设置里点「重建知识库索引」后检索会更准。",
  collection_empty: "知识库的语义索引是空的，本次只按关键词匹配。到 AI 设置里点「重建知识库索引」后检索会更准。",
  index_root_mismatch: "语义索引不是用当前知识库建的，本次只按关键词匹配。到 AI 设置里点「重建知识库索引」。",
  embedding_empty: "向量模型没有返回结果，本次只按关键词匹配。请检查 AI 设置里的向量模型。",
  exception: "向量检索调用失败，本次只按关键词匹配。请检查 AI 设置里的向量模型，或稍后重试。",
};

export function degradedNotice(r) {
  if (!r?.degraded) return null;
  const reason = r.degraded_reason || "unknown";
  if (reason === "no_semantic_candidates") return null;
  return {
    kind: "kb_degraded",
    reason,
    text: DEGRADED_NOTICES[reason] || "语义检索暂不可用，本次只按关键词匹配。可到 AI 设置检查向量模型或重建知识库索引。",
    action: "zhahui-settings",
  };
}

// 证据头部与原问答链路一致：[E3] [[slug]] 标题 > 章节；全文已在上下文里的段只给头部
function renderEvidence(ref, it, brief) {
  const head = `[${ref}] [[${it.slug}]] ${it.title || it.slug}${it.heading ? ` > ${it.heading}` : ""}`;
}

```

```

    return brief ? `${head}` (正文前面已给出, 直接沿用 ${ref}) ` :
    `${head}\n${flattenEvidenceText(it.text)}`;
  }
  ...
  const search = {
    definition: {
      name: "kb_search",
      label: "检索知识库",
      description: "检索本厂工业工程知识库 (Wiki), 返回编号证据段落 [En], 不调用大模型, 约 1 秒。回答 IE 概念、方法、标准、案例类问题时先用它; 一次没找全可以换关键词、同义词多搜几次; 段落不够时再用 kb_read 读全文。",
      promptSnippet: "检索工业工程知识库, 返回带编号的证据段落",
      parameters: Type.Object({
        query: Type.String({ description: "检索词: 具体的关键词或短语 (如「线平衡率 计算」「ECRS 原则)」, 把对话里的指代补全, 1-500 字" }),
        top_k: Type.Optional(Type.Integer({ minimum: 1, maximum: 20, description: "最多返回段落数, 默认 8" })),
      }),
    },
    async run({ query, top_k }, ctx) {
      const { signal, sessionId } = ctx;
      const q = String(query || "").trim().slice(0, 500);
      const r = await kb.post("/api/v1/retrieve", { query: q, top_k: top_k || 8 }, { signal, timeoutMs: 30000 });
      const items = (r?.items || []).filter((it) => it?.slug).map((it) => ({
        ...it,
        chunk: it.chunk_id || `${it.line_start ?? ""}-${it.line_end ?? ""}`,
        clipped: String(it.text || "").length > RETRIEVE_CLIP && String(it.text).endsWith("..."),
      }));
      const keywords = (r?.keywords || []).slice(0, 8).join("、");
      const notice = degradedNotice(r);
      const note = notice ? "\n(语义检索暂不可用, 本次只按关键词匹配, 可多换几个说法再搜)" : "";
      if (!items.length) {
        return {
          text: `知识库未检索到相关内容 (检索词: ${keywords || q})。可以换同义词或更具体的关键词再搜; 仍然没有就按「证据为空」的规则作答。${note}`,
          evidence: [],
          notice,
          summary: "未命中",
        };
      }
      const { refs, evidence } = numberItems(ctx, items);
      const blocks = items.map((it, i) => renderEvidence(refs[i].ref, it, refs[i].brief));
      const cards = topPages(items).map(({ item }) => pageCard(kb, deepRoute(item), sessionId, item.title || item.slug, { subtitle: "知识库页面" }));
      return {
        text: `检索到 ${items.length} 段知识库证据。引用时在论断句末写 [[slug#En]]:
        \n\n${blocks.join("\n\n")}${note}`,
        evidence,
        notice,
        cards,
        summary: `${evidence.length} 段证据 · ${new Set(items.map((it) => it.slug)).size} 页`,
      };
    },
  };

```

B.16 追问建议生成

回答结束后另发一次轻量请求生成三条追问建议。先去掉图片占位和引用标记，再从模型输出里容错提取第一个 JSON 数组；任何失败都返回空列表，不影响主回答。

(modules/agent/web/lib/zhihui/followups.mjs)

```
export function buildFollowupMessages({ question, answer }) {
  const q = String(question || "").trim().slice(0, MAX_QUESTION);
  const a = stripAnswerMarkup(answer).slice(0, MAX_ANSWER);
  const content = FOLLOWUP_PROMPT.replace("{question}", () => q || "(无)").replace("{answer}", ()
=> a);
  return [{ role: "user", content }];
}

// 在文本里找第一个能解析的 JSON 数组（容忍代码块、<think>、前置说明）
function firstJsonArray(text) {
  const s = String(text || "").replace(/<think>[\s\S]*?<\think>/gi,
  "").replace(/^[^\s\S]*<\think>/i, "");
  for (let start = s.indexOf("["); start >= 0; start = s.indexOf("[", start + 1)) {
    let depth = 0;
    let inStr = false;
    let esc = false;
    for (let i = start; i < s.length; i++) {
      const ch = s[i];
      if (inStr) {
        if (esc) esc = false;
        else if (ch === "\\") esc = true;
        else if (ch === "\"") inStr = false;
        continue;
      }
      if (ch === "\"") inStr = true;
      else if (ch === "[") depth++;
      else if (ch === "]" && --depth === 0) {
        try {
          const value = JSON.parse(s.slice(start, i + 1));
          if (Array.isArray(value)) return value;
        } catch {}
        break;
      }
    }
  }
  return null;
}

export function parseFollowups(text) {
  const arr = firstJsonArray(text);
  if (!arr) return [];
  const out = [];
  for (const item of arr) {
    if (typeof item !== "string") continue;
    const v = item.trim();
    if (!v || v.length > MAX_ITEM || out.includes(v)) continue;
    out.push(v);
    if (out.length >= MAX_ITEMS) break;
  }
}
```

```

    }
    return out;
  }
  ...
export async function generateFollowups(llm, { question, answer }, { fetchImpl = fetch, signal,
timeoutMs = 8000 } = {}) {
  try {
    if (!llm?.model || !llm?.baseUrl || !String(answer || "").trim()) return [];
    if (!llm.apiKey && llm.provider !== "ollama") return [];
    const { url, init } = request(llm, buildFollowupMessages({ question, answer }));
    const res = await fetchImpl(url, { ...init, signal: combineSignals(signal, timeoutMs),
redirect: "manual" });
    if (!res?.ok) return [];
    const body = await res.json().catch(() => null);
    return parseFollowups(replyText(llm, body));
  } catch {
    return [];
  }
}

```

B.17 向用户提问工具

关键信息缺失时，模型调用 `ask_user` 给出最多 4 个问题、每题 2~5 个选项。参数先清洗截断，结果带 `terminate` 结束本轮，前端把问题渲染成选项卡片，用户的选择作为下一条消息发回。
(modules/agent/web/lib/zhihui/ask-user.mjs)

```

// 信息不够时向用户提问：结果带 terminate 结束本轮，前端把 details.questions 渲染成选项卡片，
// 用户的选择作为下一条普通消息发回来。

export const ASK_USER_TOOL = "ask_user";
const MAX_QUESTIONS = 4;
const MAX_OPTIONS = 5;

const clip = (value, max) => {
  const s = typeof value === "string" ? value.replace(/\s+/g, " ").trim() : "";
  return s.length > max ? s.slice(0, max) : s;
};

/** 清洗模型给的问题：去空、截断、限数量；没有可用问题时返回空数组。 */
export function normalizeQuestions(raw) {
  if (!Array.isArray(raw)) return [];
  const out = [];
  for (const item of raw) {
    if (!item || typeof item !== "object") continue;
    const question = clip(item.question, 200);
    if (!question) continue;
    const seen = new Set();
    const options = [];
    for (const opt of Array.isArray(item.options) ? item.options : []) {
      const label = clip(typeof opt === "string" ? opt : opt?.label, 60);
      if (!label || seen.has(label)) continue;
      seen.add(label);
      const description = clip(opt?.description, 120);

```

```

    options.push(description ? { label, description } : { label });
    if (options.length >= MAX_OPTIONS) break;
  }
  const header = clip(item.header, 16);
  out.push({
    question,
    ...(header ? { header } : {}),
    options,
    multiple: item.multiple === true,
  });
  if (out.length >= MAX_QUESTIONS) break;
}
return out;
}
...
export function runAskUser(params) {
  const questions = normalizeQuestions(params?.questions);
  if (!questions.length) {
    return {
      content: [{ type: "text", text: "提问失败：没有有效的问题。请直接用文字向用户说明需要哪些信息。" }],
      details: { summary: "提问参数无效" },
    };
  }
  return {
    content: [{ type: "text", text: `已向用户提出以下问题，等待回答：\n${describe(questions)}` }],
    details: { kind: "zhihui-ask", questions, summary: `提了 ${questions.length} 个问题` },
    terminate: true,
  };
}
}

```

B.18 视频分片上传与校验

智能体把本地视频导入工时分析时，先流式计算 sha256，再按固定大小分片 PUT 上传，每片核对服务端已收字节数；完成接口返回 409 时说明资源库已有同一视频，直接提示用户不必重复导入。

(modules/agent/web/lib/zhihui/plugins/worktime/index.mjs)

```

async function uploadVideo(abs, target, { signal, onUpdate, label = "" } = {}) {
  const stat = await fs.promises.stat(abs);
  const filename = path.basename(abs);
  const mime = VIDEO_MIME[path.extname(abs).toLowerCase()];
  progress(onUpdate, `${label}正在计算文件校验值...`);
  const hash = crypto.createHash("sha256");
  for await (const chunk of fs.createReadStream(abs, { signal })) hash.update(chunk);
  const sha256 = hash.digest("hex");

  const upload = await post("/api/v1/videos/uploads", {
    filename, mime_type: mime, size_bytes: stat.size, library_node_id: target?.id ?? null,
    sha256,
  }, { signal });
  const handle = await fs.promises.open(abs, "r");
  try {
    let offset = 0;

```

```

while (offset < stat.size) {
  const length = Math.min(chunkSize, stat.size - offset);
  const buffer = Buffer.alloc(length);
  const { bytesRead } = await handle.read(buffer, 0, length, offset);
  if (!bytesRead) throw err(0, "读取文件中断");
  const res = await fetchChunk(new URL(`/api/v1/videos/uploads/${enc(upload.id)}`,
wt.backend), {
    method: "PUT",
    headers: { "Content-Type": "application/octet-stream", ...(await wt.authHeader?.()) },
    body: buffer.subarray(0, bytesRead),
    signal: signal ? AbortSignal.any([signal, AbortSignal.timeout(120000)]) :
AbortSignal.timeout(120000),
  });
  const body = await res.json().catch(() => null);
  if (!res.ok) throw err(res.status, typeof body?.detail === "string" ? body.detail : `上传失
败 HTTP ${res.status}`);
  offset += bytesRead;
  const received = unwrap(body)?.received_size;
  if (received !== undefined && received !== offset) throw err(0, `上传进度不一致（服务端
${received} 字节，本地 ${offset} 字节）`);
  progress(onUpdate, `${label}正在上传 ${Math.floor((offset / stat.size) * 100)}%`);
}
} finally {
  await handle.close();
}
}
try {
  return await post(`/api/v1/videos/uploads/${enc(upload.id)}/complete`, undefined, { signal,
timeoutMs: 300000 });
} catch (e) {
  if (e?.status === 409) throw err(409, `资源库里已经有一个和「${filename}」完全相同的视频，不用重
复导入`);
  throw e;
}
}

```

B.19 自适应视频抽帧

按视频时长自适应采样率：短视频每秒 2~4 帧，长视频按 240 帧上限摊薄，短边缩到 480 像素以控制视觉模型的输入量。帧时间戳按序号和采样率反算，供后续逐帧对齐。

(modules/worktime/backend/app/services/auto_analysis/frames.py)

```

MAX_FRAMES = 240
MAX_FPS = 4.0
MIN_FPS = 0.05
SHORT_SIDE = 480
_DURATION_RE = re.compile(r"Duration:\s*(\d+):(\d+):(\d+(?:\.\d+)?)")
...
def sample_fps(duration_ms: int, max_frames: int = MAX_FRAMES) -> float:
    """短视频 2-4 fps，长视频按总帧数上限摊薄。"""
    seconds = max(duration_ms, 1) / 1000
    return max(MIN_FPS, min(MAX_FPS, max_frames / seconds))

```

```

def extract_frames(
    video_path: Path,
    out_dir: Path,
    duration_ms: int,
    cancel: threading.Event | None = None,
    *,
    max_frames: int = MAX_FRAMES,
    short_side: int = SHORT_SIDE,
) -> list[Frame]:
    out_dir.mkdir(parents=True, exist_ok=True)
    fps = sample_fps(duration_ms, max_frames)
    # 短边缩到 short_side, 不放大; 单引号保护表达式里的逗号
    scale = (
        f"scale='if(gt(iw,ih),-2,min({short_side},iw))'"
        f":'if(gt(iw,ih),min({short_side},ih),-2)'"
    )
    cmd = [
        ffmpeg_exe(),
        "-hide_banner",
        "-loglevel",
        "error",
        "-nostdin",
        "-i",
        str(video_path),
        "-an",
        "-vf",
        f"fps={fps:.6f},{scale}",
        "-frames:v",
        str(max_frames),
        "-q:v",
        "5",
        str(out_dir / "f_%05d.jpg"),
    ]
    code, err = _run(cmd, cancel, max(120.0, duration_ms / 1000 * 2))
    files = sorted(out_dir.glob("f_*.jpg"))
    if code != 0 and not files:
        raise AutoAnalysisError(f"抽帧失败: {err.strip()[-200:] or code}")
    frames: list[Frame] = []
    for index, path in enumerate(files):
        timestamp = int(round(index * 1000 / fps))
        frames.append(Frame(timestamp_ms=min(timestamp, max(duration_ms - 1, 0)), path=path))
    if not frames:
        raise AutoAnalysisError("未能从视频中抽取画面")
    return frames

```

B.20 自动分析流水线

后台任务按抽帧、识别画面、归并要素、写入修订四个阶段推进，每步更新进度并检查取消。结果写成分析的新修订而不覆盖原有标注，失败或取消都会清理临时帧目录。
(modules/worktime/backend/app/services/auto_analysis/service.py)

```

async def _run(self, job_id: str) -> None:
    async with database.AsyncSessionLocal() as session:

```

```

        job = await session.get(AutoAnalysisJob, job_id)
        if job is None or job.status != "queued":
            return
        video = await session.get(VideoAsset, job.video_asset_id)
        job.status = "running"
        job.stage = STAGE_FRAMES
        job.progress = 1
        job.message = "正在抽取视频画面"
        await session.commit()
        mode = job.mode
        analysis_id = job.analysis_id
        video_path = _safe_media_path(self.media_dir, video.relative_path) if video else None
        video_name = video.filename if video else ""
        duration_ms = video.duration_ms if video else None

    temp_dir = self.work_dir / job_id
    try:
        if video_path is None or not video_path.is_file():
            raise AutoAnalysisError("找不到视频文件")
        vision = ai_config.vision_config()
        if vision is None:
            raise AutoAnalysisError(NOT_CONFIGURED)

        if not duration_ms:
            duration_ms = await asyncio.to_thread(frame_tools.probe_duration_ms, video_path,
self._cancel_event)
        if not duration_ms:
            raise AutoAnalysisError("无法读取视频时长")
        frames = await asyncio.to_thread(
            frame_tools.extract_frames, video_path, temp_dir, int(duration_ms),
self._cancel_event
        )
        self._check_cancel()
        await self._update(job_id, progress=10, message=f"已抽取 {len(frames)} 帧")

        records = await self._describe_frames(job_id, mode, vision, frames, int(duration_ms))
        self._check_cancel()

        await self._update(job_id, stage=STAGE_MERGE, progress=82, message="正在归并作业要素")
        hint_labels = await self._joint_hint_labels(analysis_id, mode)
        tracks, count = await self._merge(mode, vision, records, int(duration_ms), hint_labels)
        self._check_cancel()

        await self._update(job_id, stage=STAGE_WRITE, progress=94, message="正在写入新修订")
        result_id = await self._write(job_id, mode, analysis_id, video_name, tracks,
int(duration_ms))
        await self._update(
            job_id,
            status="succeeded",
            progress=100,
            result_analysis_id=result_id,
            segment_count=count,
            message=f"已生成 {count} 个作业要素",
            error=None,
        )
    except (asyncio.CancelledError, AutoAnalysisCancelled):

```

```

        await asyncio.shield(self._update(job_id, status="cancelled", message="已取消"))
    except AutoAnalysisError as error:
        await self._update(job_id, status="failed", error=str(error)[:2048], message="分析失败")
    except Exception as error: # noqa: BLE001
        logger.exception("auto analysis %s failed", job_id)
        await self._update(job_id, status="failed", error=f"自动分析出错: {error}"[:2048],
message="分析失败")
    finally:
        shutil.rmtree(temp_dir, ignore_errors=True)

```

B.21 分批识别视频画面

帧按配置的批量大小分批送入视觉模型，每帧前附时间戳，并把上一批最后一帧的记录带入以保持动作连续。单批解析失败时重试一次，仍失败记为「不清楚」；超过半数批次失败才判定整体失败。
(modules/worktime/backend/app/services/auto_analysis/service.py)

```

async def _describe_frames(
    self,
    job_id: str,
    mode: str,
    vision: dict[str, Any],
    frames: list[frame_tools.Frame],
    duration_ms: int,
) -> list[dict[str, Any]]:
    size = max(1, int(vision.get("maxFrames") or ai_config.DEFAULT_MAX_FRAMES))
    batches = [frames[index:index + size] for index in range(0, len(frames), size)]
    await self._update(job_id, stage=STAGE_VISION, progress=12, message=f"识别画面
0/{len(batches)} 批")
    records: list[dict[str, Any]] = []
    previous: dict[str, Any] | None = None
    failed = 0
    for index, batch in enumerate(batches, start=1):
        self._check_cancel()
        content: list[dict[str, Any]] = [
            {
                "type": "text",
                "text": prompts.frame_batch_text(
                    duration_ms=duration_ms,
                    batch_index=index,
                    batch_total=len(batches),
                    frame_count=len(batch),
                    previous=previous,
                    mode=mode,
                ),
            }
        ]
        for frame in batch:
            encoded = base64.b64encode(frame.path.read_bytes()).decode("ascii")
            content.append({"type": "text", "text": f"t={frame.timestamp_ms}"})
            content.append({"type": "image_url", "image_url": {"url":
f"data:image/jpeg;base64,{encoded}"}})
            messages = [{"role": "system", "content": prompts.FRAME_SYSTEM}, {"role": "user",
"content": content}]

```

```

        batch_records = await self._describe_batch(vision, messages, batch)
        if batch_records is None:
            failed += 1
            batch_records = [{"t": frame.timestamp_ms, "state": "不清楚"} for frame in batch]
        records.extend(batch_records)
        previous = batch_records[-1] if batch_records else previous
        progress = 12 + int(68 * index / len(batches))
        await self._update(job_id, progress=progress, message=f"识别画面 {index}/{len(batches)}
批")
    if failed * 2 > len(batches):
        raise AutoAnalysisError("视觉模型多数批次返回无法解析, 请确认所选模型支持图片输入")
    return records

    async def _describe_batch(
        self,
        vision: dict[str, Any],
        messages: list[dict[str, Any]],
        batch: list[frame_tools.Frame],
    ) -> list[dict[str, Any]] | None:
        for _ in range(2):
            text = await self.client.complete(vision, messages, max_tokens=VISION_MAX_TOKENS,
temperature=VISION_TEMPERATURE)
            try:
                data = parse_json(text, ("frames",))
            except AutoAnalysisError:
                continue
            items = [item for item in data.get("frames", []) if isinstance(item, dict)]
            if not items:
                continue
            by_time = {}
            for item in items:
                try:
                    by_time[int(item.get("t"))] = item
                except (TypeError, ValueError):
                    pass
            aligned = []
            for position, frame in enumerate(batch):
                item = by_time.get(frame.timestamp_ms)
                if item is None and len(items) == len(batch):
                    item = items[position]
                record = dict(item or {"state": "不清楚"})
                record["t"] = frame.timestamp_ms
                aligned.append(record)
            return aligned
        return None

```

B.22 视觉模型提示词

第一阶段让视觉模型逐帧客观记录状态、双手动作和物品, 按分析方式追加关注点; 第二阶段按模式给出归并规则, 例如模特法要求左右手共用切分点并写出分析式。
(modules/worktime/backend/app/services/auto_analysis/prompts.py)

```
FRAME_SYSTEM = """你是资深工业工程（IE）现场分析师, 正在为时间研究和动作分析做逐帧观察记录。
```

你会收到同一段作业视频按时间顺序抽出的若干帧，每帧前标注了时间戳 t （毫秒）。

逐帧客观记录以下内容，用简短中文短语：

- **left**: 左手在做什么（伸手/抓取/搬运/定位/装配/使用工具/放下/持住工件/空闲），手中是什么
- **right**: 右手，同上
- **eyes**: 视线落点（注视工件/找料/看图纸/看设备/看屏幕/离开工作区）
- **object**: 正在处理的工件、零件或工具
- **equipment**: 设备状态（运转/加工中/停止/开门/无设备）
- **body**: 身体姿态和位置（在工位站立/坐姿/弯腰/转身/走动/离开工位）
- **state**: 只能取 **working**（在做事）、**waiting**（等设备或等料）、**walking**（走动）、**searching**（找东西）、**idle**（闲置）、**absent**（人不在画面）
- **actors**: 画面里有多个人或人和设备时，按画面位置从左到右给稳定编号（作业员 1、作业员 2、设备 1），写 `[{"id","role":"作业员|设备","doing","state"}]`；只有一个人时可给空数组
- **change**: 和上一帧相比动作是否发生切换（开始伸手、抓住、放下、开始等待、开始走动等）**true/false**

规则：

1. 只写看得到的，不臆测工艺目的；看不清写「不清楚」。
2. 左右以画面中作业员自己的左右为准，看不出时按画面左右并保持前后一致。
3. 每一帧都要输出， t 原样返回；**actors** 的编号跨批次保持一致。
4. 只输出严格 JSON，不要解释，不要 markdown：

```
{"frames":[{"t":0,"left":"","right":"","eyes":"","object":"","equipment":"","body":"","state":"working","actors":[],"change":false}]}"
```

```
def frame_batch_text(
    *,
    duration_ms: int,
    batch_index: int,
    batch_total: int,
    frame_count: int,
    previous: dict[str, Any] | None,
    mode: str,
) -> str:
    lines = [
        f"视频总时长 {duration_ms / 1000:.1f} 秒。本批是第 {batch_index}/{batch_total} 批，共 {frame_count} 帧。",
    ]
    if mode in {"joint", "human_machine"}:
        lines.append("本次是联合作业分析：请务必为每个作业员和设备填写 actors。")
    if mode in {"hands", "modapts", "therblig"}:
        lines.append("本次关注双手动作：left/right 要写清楚每只手的动作和手中物品。")
    if previous:
        lines.append("上一批最后一帧的记录（用于保持编号和动作连续）：" + json.dumps(previous, ensure_ascii=False))
    lines.append("下面依次是各帧。")
    return "\n".join(lines)
```

MODAPTS_RULES = ""当前分析方式：模特排时法（MODAPTS），左右手两条轨道共用同一组切分点。1 MOD = 0.129 秒，MOD 值由系统按分析式计算。

分析式代号（数字即 MOD 数）：

- 移动：M1 手指（约 2.5cm）、M2 手腕（约 5cm）、M3 前臂（约 15cm）、M4 上臂（约 30cm）、M5 伸直手臂（约 45cm）、M7 手臂加躯干辅助
- 抓取终结：G0 触碰（不需抓握）、G1 简单抓取、G3 需要注意力的复杂抓取（小件、缠绕件、需挑选）
- 放置终结：P0 简单放置（无需对准）、P2 需一次校正或目视对准、P5 精确放置（多次校正、紧配合）
- 其他：L1 重量修正（每增加约 4kg）、E2 眼动或对焦、R2 校正（改握、调整位置）、D3 判断、A4 施压（压紧、拧紧到位）、C4 摇转（每圈）、F3 脚踏、W5 步行（每步）、B17 弯腰并起身、S30 坐下并站起

每行字段:

- left / right 各含:

activity: 该手动作描述

mod_formula: 分析式, 按动作顺序把代号连写, 一般是「移动+终结」, 如「伸手抓螺丝」= M3G1, 「放入孔位」= M3P2, 「拿起后转交另一只手」= M2G1

mod_count: 同一分析式连续重复次数 (如摇转 3 圈写 C4 次数 3), 默认 1

规则:

1. 任一手动作转换就切一行; 另一只手若在持住或空闲, **activity** 写「持住」或「空闲」, **mod_formula** 留空字符串。
2. 两手同时动作时各自填写分析式, 系统取较大一侧, 不要人为相加。
3. 只用上面列出的代号, 不要自创。

输出:

```
{ "rows": [ { "start_ms": 0, "end_ms": 800, "left": { "activity": "伸手抓外壳",  
"mod_formula": "M4G1", "mod_count": 1 }, "right": { "activity": "空闲", "mod_formula": "", "mod_count": 1 } } ] }
```

B.23 模型接口调用与重试

兼容 OpenAI 格式的最小客户端: 自动补全 /v1/chat/completions, 超时、连接错误和 429/5xx 按退避重试, 鉴权失败立即报错提示检查 Key, 并兼容把 content 拆成片段返回的服务。

(modules/worktime/backend/app/services/auto_analysis/llm.py)

```
def completions_url(base_url: str) -> str:
    """https://x.com -> /v1/chat/completions; 已带 /v1、/v3 等版本段的直接接 /chat/completions。"""
    root = (base_url or "").strip().rstrip("/")
    if not root:
        raise AutoAnalysisError("AI 设置里缺少接口地址")
    if root.endswith("/chat/completions"):
        return root
    if _VERSION_SUFFIX.search(root):
        return f"{root}/chat/completions"
    return f"{root}/v1/chat/completions"
...
async def complete(
    self,
    endpoint: dict[str, Any],
    messages: list[dict[str, Any]],
    *,
    max_tokens: int,
    temperature: float,
) -> str:
    url = completions_url(str(endpoint.get("baseUrl") or ""))
    api_key = str(endpoint.get("apiKey") or "")
    if not api_key:
        raise AutoAnalysisError("AI 设置里缺少 API Key")
    body = {
        "model": endpoint["model"],
        "messages": messages,
        "max_tokens": int(max_tokens),
        "temperature": float(temperature),
        "stream": False,
    }
    headers = {"Authorization": f"Bearer {api_key}", "Content-Type": "application/json"}
    last_error = ""
    async with httpx.AsyncClient(timeout=self.timeout, transport=self.transport) as client:
```

```

    for attempt in range(self.retries + 1):
        if attempt:
            await asyncio.sleep(self.backoff_s * attempt)
        try:
            response = await client.post(url, json=body, headers=headers)
        except httpx.TimeoutException:
            last_error = "模型接口超时"
            continue
        except httpx.TransportError as error:
            last_error = f"无法连接模型接口: {type(error).__name__}"
            continue
        if response.status_code in (401, 403):
            raise AutoAnalysisError(f"模型接口鉴权失败 ({response.status_code}), 请检查 API Key")

        if response.status_code in RETRY_STATUS:
            last_error = f"模型接口返回 {response.status_code}: {response.text[:200]}"
            continue
        if response.status_code >= 400:
            raise AutoAnalysisError(f"模型接口返回 {response.status_code}: {response.text[:300]}")
        try:
            payload = response.json()
            content = payload["choices"][0]["message"]["content"]
        except (ValueError, KeyError, IndexError, TypeError) as error:
            raise AutoAnalysisError("模型接口返回格式无法识别") from error
        if isinstance(content, list): # 部分服务把 content 拆成片段
            content = "".join(str(part.get("text", "")) for part in content if
                               isinstance(part, dict))
        return str(content or "")
    raise AutoAnalysisError(last_error or "模型接口调用失败")

```

B.24 时间轴片段校验

保存分析前统一校验：片段模式和轨道一致、时长为正且不超出视频、无效区间必须落在片段内，同一轨道上的片段不得重叠。联合作业最多 6 名作业员。
(modules/worktime/backend/app/services/timeline.py)

```

def _validate_segment(mode: str, track: Track, segment: Segment, video_duration_ms: int | None) -> None:
    if segment.mode != mode:
        raise TimelineValidationError("segment mode must match the analysis mode")
    if segment.track_key != track.track_key:
        raise TimelineValidationError("segment track key must match its track")

    start_ms = _require_milliseconds(segment.start_ms, "start_ms")
    end_ms = _require_milliseconds(segment.end_ms, "end_ms")
    if start_ms < 0 or end_ms <= start_ms:
        raise TimelineValidationError("segment duration must be positive")
    if video_duration_ms is not None and end_ms > video_duration_ms:
        raise TimelineValidationError("segment exceeds video duration")
    if segment.category not in VALID_CATEGORIES:
        raise TimelineValidationError("unsupported segment category")
    if segment.activity_type is not None and segment.activity_type not in VALID_ACTIVITY_TYPES:

```

```

        raise TimelineValidationError("unsupported activity type")
    ...
    invalid_start_ms = segment.invalid_start_ms
    invalid_end_ms = segment.invalid_end_ms
    if (invalid_start_ms is None) != (invalid_end_ms is None):
        raise TimelineValidationError("invalid interval requires both boundaries")
    if invalid_start_ms is not None and invalid_end_ms is not None:
        invalid_start_ms = _require_milliseconds(invalid_start_ms, "invalid_start_ms")
        invalid_end_ms = _require_milliseconds(invalid_end_ms, "invalid_end_ms")
        if invalid_start_ms < start_ms or invalid_end_ms > end_ms or invalid_end_ms <=
invalid_start_ms:
            raise TimelineValidationError("invalid interval must be inside its segment")

def normalize_segments(mode: str, tracks: Sequence[Track], video_duration_ms: int | None = None) ->
list[Track]:
    """Validate editable tracks and return independent objects in timeline order."""
    if mode not in VALID_MODES:
        raise TimelineValidationError("unsupported analysis mode")
    if video_duration_ms is not None:
        video_duration_ms = _require_milliseconds(video_duration_ms, "video_duration_ms")
        if video_duration_ms < 0:
            raise TimelineValidationError("video_duration_ms cannot be negative")

    source_tracks = sorted(list(tracks), key=lambda track: (track.sort_order, track.track_key))
    if mode == "joint":
        operator_keys = [track.track_key for track in source_tracks if
track.track_key.startswith("operator_")]
        if len(source_tracks) > MAX_JOINT_OPERATORS or len(operator_keys) > MAX_JOINT_OPERATORS:
            raise TimelineValidationError("joint analysis supports at most 6 operators")
        seen_track_keys: set[str] = set()
        normalized_tracks: list[Track] = []
        for track in source_tracks:
            if not track.track_key or track.track_key in seen_track_keys:
                raise TimelineValidationError("track keys must be unique and non-empty")
            seen_track_keys.add(track.track_key)

            for segment in track.segments:
                _validate_segment(mode, track, segment, video_duration_ms)
            ordered_segments = sorted(track.segments, key=lambda segment: (segment.start_ms,
segment.end_ms, segment.activity))
            previous_end_ms = 0
            for segment in ordered_segments:
                if segment.start_ms < previous_end_ms:
                    raise TimelineValidationError("segments on a track cannot overlap")
                previous_end_ms = segment.end_ms

```

B.25 模特排时法计算

分析式中的数字相加即 MOD 数，乘以次数得单手 MOD，一行取左右手较大者；正常时间 = MOD × 0.129 秒。标准时间按 Python 的取偶舍入，保证与后端导出结果一致。
(modules/worktime/frontend/src/features/workbench/modapts.ts)

```

export const MOD_UNIT_SECONDS = 0.129

export function parseModFormula(formula: string | null | undefined): number {
  if (!formula) return 0
  const matches = formula.match(/\d+/g)
  if (!matches) return 0
  return matches.reduce((sum, token) => sum + Number(token), 0)
}

export function handModValue(formula: string | null | undefined, count: number | null | undefined):
number {
  const times = Number.isFinite(count) && Number(count) > 0 ? Math.round(Number(count)) : 0
  return parseModFormula(formula) * times
}

export function rowModValue(left: number, right: number): number {
  return Math.max(left, right)
}

export function modUnitSeconds(value: number | null | undefined): number {
  if (!Number.isFinite(value) || Number(value) <= 0) return MOD_UNIT_SECONDS
  return Number(value)
}

export function normalTimeMs(modValue: number, unitSeconds = MOD_UNIT_SECONDS): number {
  return Math.round(modValue * modUnitSeconds(unitSeconds) * 1_000)
}

function roundLikePython(value: number): number {
  const lower = Math.floor(value)
  const fraction = value - lower
  if (fraction === 0.5) return lower % 2 === 0 ? lower : lower + 1
  return Math.round(value)
}

export function standardTimeMs(normalMs: number, allowance: number): number {
  return roundLikePython(normalMs * (1 + allowance))
}

export interface ModaptsRowTotal {
  modValue: number
  unitSeconds: number
  normalMs: number
  standardMs: number
}

...
const shared = left ?? right
if (!shared) return null
const resolvedUnit = modUnitSeconds(unitSeconds)
const modValue = rowModValue(handModValue(left?.mod_formula, left?.mod_count),
handModValue(right?.mod_formula, right?.mod_count))
const normalMs = normalTimeMs(modValue, resolvedUnit)
return {
  modValue,
  unitSeconds: resolvedUnit,
  normalMs,

```

```

        standardMs: standardTimeMs(normalMs, shared.allowance),
    }
}

export function sumModaptsTotals(
    tracks: AnalysisTrack[],
    unitSeconds = MOD_UNIT_SECONDS,
): { modValue: number; normalMs: number; standardMs: number } {
    const left = tracks.find((track) => track.track_key === 'left_hand')
    const right = tracks.find((track) => track.track_key === 'right_hand')
    const length = Math.max(left?.segments.length ?? 0, right?.segments.length ?? 0)
    let modValue = 0
    let normalMs = 0
    let standardMs = 0
    for (let index = 0; index < length; index += 1) {
        const row = rowModaptsTotal(left?.segments[index], right?.segments[index], unitSeconds)
        if (!row) continue
        modValue += row.modValue
        normalMs += row.normalMs
        standardMs += row.standardMs
    }
    return { modValue, normalMs, standardMs }
}

```

B.26 前后端计算口径对齐

后端在生成报告和供智能体查询时复算工时指标，逐一复刻前端的 `Math.round`、`toFixed` 和取偶舍入，使页面、导出文件和对话里给出的数字完全相同。
(modules/worktime/backend/app/services/insights.py)

```

def js_round(value: float) -> int:
    """JS Math.round: 最近整数，正好 .5 时取向 +∞ 的那个。"""
    lower = math.floor(value)
    return int(lower + 1 if value - lower >= 0.5 else lower)

def py_round(value: float) -> int:
    """前端 roundLikePython: 等价于 Python 内置 round (.5 取偶)。"""
    return int(round(value))

def fixed(value: float, digits: int = 2) -> float:
    """JS Number.prototype.toFixed 的数值: 按浮点精确值四舍五入 (远离 0)。"""
    quantum = Decimal(1).scaleb(-digits)
    return float(Decimal(value).quantize(quantum, rounding=ROUND_HALF_UP))

def seconds(ms: float | None) -> float | None:
    """毫秒 → 秒，两位小数，与页面 formatSeconds (toFixed(2)) 一致。"""
    if ms is None:
        return None
    return fixed(ms / 1_000, 2)

```

```

def _js_hundredths(value: float) -> float:
    """sopModel.roundHundredths: Math.round(value * 100) / 100. """
    return js_round(value * 100) / 100
...
def segment_times(segment: Any) -> dict[str, int]:
    """tableRows.segmentTimes: 实测、无效、有效; 标准 = 浪费按 0, 否则 有效 × 评定 × (1+宽放)。"""
    seg = segment if isinstance(segment, dict) and "invalid_start_ms" in segment else
    _segment_dict(segment)
    measured = max(0, seg["end_ms"] - seg["start_ms"])
    invalid = segment_invalid_ms(seg)
    valid = max(0, measured - invalid)
    standard = 0 if seg["category"] == "waste" else py_round(valid * seg["rating"] * (1 +
seg["allowance"]))
    return {"measured_ms": measured, "invalid_ms": invalid, "valid_ms": valid, "standard_ms":
standard}

def parse_mod_formula(formula: str | None) -> int:
    """分析式里所有连续数字相加, M3G1 = 4. """
    if not formula:
        return 0
    return sum(int(token) for token in _DIGITS.findall(formula))

def hand_mod_value(formula: str | None, count: Any) -> int:
    try:
        number = float(count)
    except (TypeError, ValueError):
        number = 0.0
    times = js_round(number) if math.isfinite(number) and number > 0 else 0
    return parse_mod_formula(formula) * times

def row_modapts_total(left: dict | None, right: dict | None, unit_seconds: float) -> dict[str, Any]
| None:
    """modapts.rowModaptsTotal: MOD = max(左, 右); 正常 = MOD × 单位; 标准 = 正常 × (1+放宽)。"""
    shared = left or right
    if not shared:
        return None
    unit = mod_unit_seconds(unit_seconds)
    mod_value = max(
        hand_mod_value(left["mod_formula"] if left else None, left["mod_count"] if left else None),
        hand_mod_value(right["mod_formula"] if right else None, right["mod_count"] if right else
None),
    )
    normal_ms = js_round(mod_value * unit * 1_000)
    return {
        "mod_value": mod_value,
        "normal_ms": normal_ms,
        "standard_ms": py_round(normal_ms * (1 + shared["allowance"])),
        "allowance": shared["allowance"],
    }
}

```

B.27 Wiki 页面切块

按二级标题切分章节，超长章节再按段落贪心合并到 300~600 字，不切开单个段落。每块记录行号区间用于深链定位，向量化文本前缀页面标题、章节名和别名以提高召回。

(modules/kgqa/backend/app/services/query/chunker.py)

```
def _pack_blocks(
    lines: list[str], blocks: list[tuple[int, int]]
) -> list[tuple[int, int]]:
    """贪心合并段落块到 300-600 字目标区间,不切开单个段落。"""
    packed: list[tuple[int, int]] = []
    cur_start: int | None = None
    cur_end = 0
    cur_len = 0
    for bs, be in blocks:
        block_len = sum(len(lines[i - 1]) for i in range(bs, be + 1))
        if cur_start is None:
            cur_start, cur_end, cur_len = bs, be, block_len
            continue
        if cur_len >= CHUNK_TARGET_MIN and cur_len + block_len > CHUNK_TARGET_MAX:
            packed.append((cur_start, cur_end))
            cur_start, cur_end, cur_len = bs, be, block_len
        else:
            cur_end = be
            cur_len += block_len
    if cur_start is not None:
        packed.append((cur_start, cur_end))
    return packed
...
def chunk_page(path: Path) -> list[dict]:
    """解析并切块一个 wiki 页面。

    返回 [{chunk_id, slug, title, kind, heading, text, line_start, line_end,
            embed_text, content_hash, chunk_index}],空页返回 []。
    """
    path = Path(path)
    try:
        fm, body = parse_text(path.read_text(encoding="utf-8"))
    except Exception:
        return []
    slug = path.stem
    title = str(fm.get("title") or slug)
    kind = str(fm.get("kind") or "")
    aliases = " ".join(str(a) for a in (fm.get("aliases") or []))
    lines = body.splitlines()
    if not lines:
        return []

    # 1) 章节区间
    spans: list[tuple[str, int, int]] = []
    for heading, s, e in _split_sections(lines):
        if not any(lines[i - 1].strip() for i in range(s, e + 1)):
            continue # 纯空白章节
        if _section_text_len(lines, s, e) <= SECTION_SPLIT_THRESHOLD:
```

```

        spans.append((heading, s, e))
        continue
# 2) 超长章节按段落块再切;标题行归入第一个子块的区间
content_start = s + 1 if heading and lines[s - 1].startswith("## ") else s
blocks = _split_paragraph_blocks(lines, content_start, e)
packed = _pack_blocks(lines, blocks)
for i, (bs, be) in enumerate(packed):
    span_start = s if i == 0 else bs
    spans.append((heading, span_start, be))

# 3) 生成 chunk 记录
chunks: list[dict] = []
for idx, (heading, s, e) in enumerate(spans):
    text_lines = [
        lines[i - 1]
        for i in range(s, e + 1)
        if not lines[i - 1].startswith("## ")
    ]
    text = "\n".join(text_lines).strip()
    if not text:
        # 章节只有标题没有正文:退化用标题行本身,保证行号可映射
        text = lines[s - 1].strip()
    head_label = heading or "引言"
    embed_text = f"{title} · {head_label}\n{text}"
    if idx == 0 and aliases:
        embed_text = f"{title} · {head_label}\n 别名: {aliases}\n{text}"
    chunks.append(
        {
            "chunk_id": f"{slug}::{idx:02d}",
            "slug": slug,
            "title": title,
            "kind": kind,
            "heading": heading,
            "text": text,
            "line_start": s,
            "line_end": e,
            "chunk_index": idx,
            "embed_text": embed_text,
            "content_hash": _hash(embed_text),
        }
    )
)
return chunks

```

B.28 增量向量索引同步

以页面级内容哈希判断是否变化,只对新增或修改的页面重新向量化,先删旧块再写新块以免留下孤儿记录;磁盘上已删除的页面同步清理,最后写入索引标记并让检索缓存失效。

(modules/kgqa/backend/app/services/query/vector_index.py)

```

async def sync_query_vector_index(
    *,
    wiki_root: Path,
    batch_size: int = 32,

```

```

force: bool = False,
progress: Callable[[dict], None] | None = None,
) -> dict:
    """Upsert chunks of new/changed pages and remove stale rows for one Wiki root."""
    wiki_root = Path(wiki_root).resolve()
    root_id = str(wiki_root)

    # slug -> (page_hash, chunks)
    pages: dict[str, tuple[str, list[dict]]] = {}
    for path in iter_wiki_pages(wiki_root):
        if _is_generated_page(path, wiki_root):
            continue
        chunks = chunk_page(path)
        if chunks:
            pages[path.stem] = (page_content_hash(chunks), chunks)

    store = WikiVectorStore(
        persist_dir=settings.chroma_persist_dir,
        host=settings.chroma_host,
        port=settings.chroma_port,
        collection_name=COLLECTION_NAME,
    )
    if force:
        await store.delete_where({"index_root": root_id})
        existing: dict[str, dict] = {}
    else:
        existing = await store.list_records(where={"index_root": root_id})
    # 现存记录按 slug 分组: slug -> {"page_hash": str, "ids": [chunk_id]}
    existing_pages: dict[str, dict] = {}
    for chunk_id, metadata in existing.items():
        slug = str(metadata.get("slug") or chunk_id.split("::")[0])
        grp = existing_pages.setdefault(slug, {"page_hash": None, "ids": []})
        grp["ids"].append(chunk_id)
        if grp["page_hash"] is None:
            grp["page_hash"] = metadata.get("page_hash")

    changed = [
        slug
        for slug, (page_hash, _) in pages.items()
        if existing_pages.get(slug, {}).get("page_hash") != page_hash
    ]
    stale_slugs = sorted(set(existing_pages) - set(pages))
    total_chunks = sum(len(chunks) for _, chunks in pages.values())
    pending_chunks = sum(len(pages[slug][1]) for slug in changed)
    ...
    provider = get_embedding_provider()
    upserted_chunks = 0
    for page_i, slug in enumerate(changed, start=1):
        page_hash, chunks = pages[slug]
        # 先删该页旧 chunk(数量/边界可能变化,直接 upsert 会留孤儿)
        old_ids = existing_pages.get(slug, {}).get("ids") or []
        if old_ids:
            await store.delete(old_ids)
        for offset in range(0, len(chunks), max(1, batch_size)):
            batch = chunks[offset : offset + max(1, batch_size)]
            embeddings = await provider.embed([c["embed_text"] for c in batch])

```

```

        await store.upsert(
            [c["chunk_id"] for c in batch],
            embeddings,
            [_chunk_metadata(c, page_hash, root_id) for c in batch],
            documents=[c["text"] for c in batch],
        )
        upserted_chunks += len(batch)
    ...
    stale_ids = [
        chunk_id for slug in stale_slugs for chunk_id in existing_pages[slug]["ids"]
    ]
    await store.delete(stale_ids)

    _write_marker(wiki_root, len(pages), total_chunks)
    ...
    return {
        "page_count": len(pages),
        "chunk_count": total_chunks,
        "upserted": len(changed),
        "upserted_chunks": upserted_chunks,
        "deleted": len(stale_slugs),
        "unchanged": len(pages) - len(changed),
        "force": force,
    }

```

B.29 别名扩展检索词

基础关键词与领域词取并集后，问题命中任一别名时把整组同义词追加到检索词。含英文或数字的别名按单词边界匹配，避免短缩写误中更长的单词；纯中文别名按子串匹配。
(modules/kgqa/backend/app/services/query/alias_expand.py)

```

@lru_cache(maxsize=4096)
def _member_pattern(member: str) -> re.Pattern[str] | None:
    """别名的边界感知正则;纯中文别名返回 None(走子串匹配)。"""
    if not _ASCII_ALNUM.search(member):
        return None
    prefix = r"(?![A-Za-z0-9])" if _ASCII_ALNUM.match(member[0]) else ""
    suffix = r"(?![A-Za-z0-9])" if _ASCII_ALNUM.match(member[-1]) else ""
    return re.compile(prefix + re.escape(member) + suffix)

def _member_matches(member: str, question: str) -> bool:
    pattern = _member_pattern(member)
    if pattern is None:
        return member in question
    return pattern.search(question) is not None

def expand(question: str, *, alias_groups: list[list[str]]) -> tuple[list[str], list[str]]:
    """返回 (expanded_keywords, hit_aliases)。

    - 基础关键词来自 basic_keywords(question)
    - 对于每个别名组,只要问题中包含任一成员 → 整组追加(去重保序,原序在前)
    """

```

```

- hit_aliases 列出"命中的别名组的 canonical(group[0])"
"""

# basic_keywords 把连续中文抓成整段,问题主语会被整句吞掉—"工时研究在智能制造
# 背景下的演进与局限"只得到 1 个整句 token,"工时研究"从未进过检索。
# meaningful_keywords 能切出领域词,但它是评分侧的过滤器,会丢掉超长 token
# (同一句里的"精益生产""六西格玛"就这么没的),所以两者取并集而不是替换。
base = basic_keywords(question)
seen: set[str] = set(base)
out: list[str] = list(base)
for term in meaningful_keywords(question):
    if term and term not in seen:
        seen.add(term)
        out.append(term)
hit_canonicals: list[str] = []

for group in alias_groups:
    if not group:
        continue
    if any(member and _member_matches(member, question) for member in group):
        canonical = group[0]
        if canonical not in hit_canonicals:
            hit_canonicals.append(canonical)
        for member in group:
            if member and member not in seen:
                seen.add(member)
                out.append(member)
return out, hit_canonicals

```

B.30 入库隐私扫描

资料入库前逐行扫描手机号、身份证号、API Key、URL 令牌等敏感信息，命中片段掩码后返回。身份证号、API Key、URL 令牌为阻断级，必须修改后再传；手机号、邮箱、银行卡为提醒级，用户确认后放行。（modules/kgqa/backend/app/services/ingestion/privacy_check.py）

```

PATTERNS: list[tuple[str, re.Pattern[str], str, str]] = [
    (
        "phone",
        re.compile(r"(?<!\d)1[3-9]\d{9}(?!\\d)"),
        "warn",
        "中国大陆手机号",
    ),
    (
        "id_card",
        re.compile(
            r"(?<!\d)[1-9]\d{5}(18|19|20)\d{2}(\d[1-9]|1[0-2])(\d[1-9]|1[2]\d|3[01])\d{3}[\dXx](?!\\d)"
        ),
        "block",
        "中国大陆身份证号",
    ),
    (
        "email",
        re.compile(r"[\w.+-]+@[\\w-]+\\. [\\w.-]+"),
    ),
]

```

```

        "warn",
        "邮箱地址",
    ),
    (
        "api_key",
        re.compile(r"(sk-|gsk-|xai-|hf-|ghp-|github_pat-)[A-Za-z0-9_-]{16,})",
        "block",
        "常见 API key 前缀",
    ),
    (
        "bank_card",
        re.compile(r"(?<!\d)\d{16,19}(?!\\d)"),
        "warn",
        "疑似银行卡号(易误报)",
    ),
    (
        "url_token",
        re.compile(r"[?&](token|access_token|api_key)=[A-Za-z0-9_-]{8,})",
        "block",
        "URL 中包含 token",
    ),
]

def _mask(snippet: str, *, keep_head: int = 3, keep_tail: int = 2) -> str:
    """对 snippet 做掩码,保留头尾,中间替换为 *。"""
    if len(snippet) <= keep_head + keep_tail:
        return "*" * len(snippet)
    return snippet[:keep_head] + "*" * (len(snippet) - keep_head - keep_tail) + snippet[-keep_tail:]

def scan(content: str) -> dict:
    ...
    hits: list[dict] = []
    if not content:
        return {"ok": True, "hits": [], "block_count": 0, "warn_count": 0}

    lines = content.splitlines()
    for line_no, line in enumerate(lines, start=1):
        for category, pat, level, label in PATTERNS:
            for m in pat.finditer(line):
                hits.append({
                    "line": line_no,
                    "category": category,
                    "snippet": _mask(m.group(0)),
                    "level": level,
                    "label": label,
                })

    block_count = sum(1 for h in hits if h["level"] == "block")
    warn_count = sum(1 for h in hits if h["level"] == "warn")
    return {
        "ok": block_count == 0,
        "hits": hits,
        "block_count": block_count,
    }

```

```
"warn_count": warn_count,  
}
```

B.31 图表生成缓存与并发

每类图表模板对应一个 generator.py，按文件修改时间缓存已加载模块；出图结果以模板和参数哈希为键做 TTL 缓存，信号量限制并发，matplotlib 绘制放到线程中执行，不阻塞事件循环。
(modules/zhihuifu/backend/app/services/chart_generator.py)

```
def _cache_key(template_id: str, params: dict) -> str:  
    payload = json.dumps(params, ensure_ascii=False, sort_keys=True, default=str)  
    digest = hashlib.sha256(payload.encode("utf-8")).hexdigest()  
    return f"{template_id}:{digest}"  
  
def _get_cached(key: str) -> dict | None:  
    if settings.CHART_GENERATION_CACHE_TTL_SECONDS <= 0:  
        return None  
    item = _cache.get(key)  
    if not item:  
        return None  
    expires_at, value = item  
    if expires_at < time.monotonic():  
        _cache.pop(key, None)  
        return None  
    _cache.move_to_end(key)  
    return value.copy()  
  
def _set_cached(key: str, value: dict) -> None:  
    ttl = settings.CHART_GENERATION_CACHE_TTL_SECONDS  
    if ttl <= 0 or settings.CHART_GENERATION_CACHE_MAX_ITEMS <= 0:  
        return  
    _cache[key] = (time.monotonic() + ttl, value.copy())  
    _cache.move_to_end(key)  
    while len(_cache) > settings.CHART_GENERATION_CACHE_MAX_ITEMS:  
        _cache.popitem(last=False)  
    ...  
  
def _load_generator(templates_dir: Path, template_id: str):  
    gen_path = _generator_path(templates_dir, template_id)  
    mtime = gen_path.stat().st_mtime  
    hit = _modules.get(template_id)  
    if hit and hit[0] == mtime:  
        return hit[1]  
    spec = importlib.util.spec_from_file_location(f"chart_gen_{template_id}", str(gen_path))  
    if spec is None or spec.loader is None:  
        raise FileNotFoundError(f"generator.py not found for template: {template_id}")  
    mod = importlib.util.module_from_spec(spec)  
    spec.loader.exec_module(mod)  
    _modules[template_id] = (mtime, mod)  
    return mod
```

```

def _run_generator(template_id: str, params: dict, fmt: str, dpi: int | None) -> dict:
    templates_dir = Path(settings.CHART_TEMPLATES_DIR).resolve()
    _ensure_templates_path(templates_dir)
    import _utils # chart-templates/_utils.py

    mod = _load_generator(templates_dir, template_id)
    token = _utils.set_render_options(fmt, dpi)
    try:
        result = mod.generate(params)
    finally:
        _utils.reset_render_options(token)
    return {"image_base64": result["image_base64"], "summary": result.get("summary", {})}

async def generate_chart(template_id: str, params: dict, fmt: str = "png", dpi: int | None = None)
-> dict:
    if fmt not in EXPORT_FORMATS:
        raise ValueError(f"不支持的导出格式: {fmt}")
    if dpi is not None:
        dpi = max(72, min(int(dpi), 600))
    key = f"_{cache_key(template_id, params)}:{fmt}:{dpi or 0}"
    cached = _get_cached(key)
    if cached is not None:
        return cached

    async with _generate_semaphore:
        cached = _get_cached(key)
        if cached is not None:
            return cached
        result = await asyncio.to_thread(_run_generator, template_id, params, fmt, dpi)
        _set_cached(key, result)
        return result

```

B.32 draw.io 文件自动同步

智能体写出的 .drawio 文件自动存入知绘图「我的图表」，用文件路径到图表 ID 的映射保证同一文件再改只更新原图。原图被用户删除时重新新建，同步失败把原因告诉模型以便重写文件。

(modules/agent/web/lib/zhihui/plugins/zhihuitu/index.mjs)

```

// .drawio 文件 → 「我的图表」 chartId: 同一文件再改只更新那张图，历史记录留在一处
const DRAWIO_MAP_FILE = "drawio-charts.json";
const fileKey = (p) => (process.platform === "win32" ? path.resolve(p).toLowerCase() :
path.resolve(p));
...
export function isDrawioPath(p) {
    return typeof p === "string" && path.extname(p).toLowerCase() === ".drawio";
}

/** 把 .drawio 文件或 XML 存进「我的图表」：文件已导入过就 PUT 原图（保留用户改过的标题），否则新建。返回
{chartId, title, file, created} */
export async function syncDrawioChart(zt, cfg, { file, xml, title, cwd, signal } = {}) {
    const call = async (method, url, opts) => unwrapEnvelope("zhihuitu", await zt.json(method, url,
opts));

```

```

const input = await readDrawioInput({ file, xml, cfg, cwd });
const explicit = String(title || "").trim().slice(0, 60);
const name = explicit || input.fallbackTitle;
const key = input.file && cfg.dataDir ? fileKey(input.file) : null;
const map = key ? readDrawioMap(cfg) : {};
const known = key ? Number(map[key]?.chartId) : 0;
if (known) {
  try {
    await call("PUT", `/api/v1/charts/${known}`, { body: { ...(explicit ? { title: explicit } : {}), chartData: { xml: input.content } }, signal });
    return { chartId: known, title: explicit || map[key].title || name, file: input.file || null, created: false };
  } catch (e) {
    // 用户在智绘图里删了这张图：重新建一张
    if (!(e instanceof ServiceError) || e.status !== 404) throw e;
  }
}
const saved = await call("POST", "/api/v1/charts", { body: { title: name, templateId: DRAWIO_TEMPLATE, chartData: { xml: input.content } }, signal });
if (key) {
  const next = readDrawioMap(cfg);
  next[key] = { chartId: saved.id, title: name };
  writeDrawioMap(cfg, next);
}
return { chartId: saved.id, title: name, file: input.file || null, created: true };
}

/** 回答里的 draw.io 入口：drawio 字段带文件路径时前端可在右侧预览 */
export function drawioCard(zt, sessionId, { chartId, title, file }) {
  return pageCard(zt, editorRoute("drawio", chartId), sessionId, title, { subtitle: "draw.io 图", drawio: file || "" });
}

/** 命令/写文件产生的 .drawio 自动同步进「我的图表」；返回给前端的卡片和给模型的说明 */
export async function autoSyncDrawioFiles(zt, cfg, files, { sessionId, signal } = {}) {
  const cards = [];
  const notes = [];
  for (const f of files || []) {
    if (f.status === "deleted" || !isDrawioPath(f.path)) continue;
    try {
      const r = await syncDrawioChart(zt, cfg, { file: f.path, signal });
      cards.push(drawioCard(zt, sessionId, r));
      notes.push(`${path.basename(f.path)} 已自动${r.created ? "保存" : "更新"}到知绘图「我的图表」(chartId=${r.chartId})，回答下方有预览和打开编辑器的入口，不用再调 drawio_save。`);
    } catch (e) {
      if (signal?.aborted) throw e;
      notes.push(`${path.basename(f.path)} 同步到知绘图失败：${e?.message || e}。检查 XML 后重写文件。`);
    }
  }
  return { cards, notes };
}

```