Vol. 4, No. 4 Aug. 2026
DOI: https://10.58244/rpiei.264012

基于边缘智能的工程约束下实时图像拼接技术实践与思考

白佳谕 [1,a,*]
[1] 西安工业大学，陕西，西安，中国
[a] 1959825177@qq.com

摘 要：本文以本科生科研项目“基于边缘智能的城市安防实时街景图像拼接技术研究”为背景，梳理项目实施过程中遇到的工程问题。文章不侧重算法性能分析，而围绕 CUDA 依赖冲突、Python 版本兼容、开源库适配和设备限制等影响开发进度的关键问题，探讨其产生原因及解决方法。结合 UDIS++ 双阶段模型在 CPU 边缘设备上的优化实践，进一步总结反向问题求解、隐性假设分析等工程方法，并反思传统工科教学中重理论、轻实践的问题。本文还提出了以问题拆解、试错验证、复盘巩固为核心的三阶实践教学方法，并尝试构建涵盖环境工程能力、版本管理能力、异常处理能力的工科基础能力体系，旨在为信息时代新工科人才的工程素养培养提供一份可参考的实践样本。

关键词：边缘智能；图像拼接；UDIS++模型；工程实践；工科教育改革

基金信息：西安工业大学国家级大学生创新创业训练项目（202510702022）。

Practice and Reflections on Real-time Image Stitching Technology Under Engineering Constraints Based on Edge Intelligence

Jiayu Bai [1,a,*]
[1] Xi’an Technological University, Xi’an, Shaanxi, China
[a] 1959825177@qq.com

Abstract: In this paper, the main problems in the implementation of the project are summarized and analyzed based on the undergraduate research project " Research on real-time street scene image mosaic technology of urban security based on edge intelligence. " This paper does not focus on the performance of the algorithm, but focuses on the engineering problems that affect the progress of system development. Combined with the optimization practice of UDIS + + two-stage model on CPU edge devices, this paper summarizes engineering analysis methods such as reverse problem solving and implicit hypothesis mining. At the same time, the problems of emphasizing theory, neglecting practice, emphasizing results, neglecting process and ignoring practical constraints in traditional engineering teaching are considered. This paper also proposes a three-stage practical teaching method with problem disassembly, trial and error verification, and re-disk consolidation as the core, and tries to construct an engineering basic ability system covering environmental engineering ability, version

management ability, and exception handling ability. The aim is to provide a practical sample for the engineering literacy training of new engineering talents in the information age.

Keywords: Edge Intelligence; Image Stitching; UDIS++ Model; Engineering Practice; Engineering Education Reform

1 引言

最近这些年国内智慧城市建设推进得很快，城市公共安全防控的思路也随之转变，从过去单点、分散的摄像头监控模式，逐步转向对全域场景的连续感知。但传统的集中式云端图像处理架构，面对海量高清视频数据时存在若干难以回避的问题，网络延迟偏高、带宽成本居高不下，数据集中传输还伴随隐私泄露的风险。这些短板让城市安防对毫秒级响应和高可靠性的需求很难真正落地。边缘智能技术的核心思路是把计算任务下沉到数据采集端的边缘设备上，使数据可以在本地完成处理，并且能够实时得到结果，这为解决上述痛点提供了一条可行的方法[1][2]。

该项目需要完成一套基于 UDIS++深度学习模型的轻量级实时街景拼接系统，核心需求就是摆脱对 GPU 硬件的依赖，使普通 CPU 边缘设备也能够完成多摄像头全景图像的生成，最终服务于低成本、广覆盖的城市安防监控网络建设[3][4]。

项目刚开始时，预计主要工作会集中在算法优化以及模型改进方面，而代码运行以及系统部署只是后续得以实现的过程中需要开展的常规工作。然而在实际开展开发工作后发现，大量时间都花费在进行环境配置、程序适配等相关问题上。这些问题单独来看并不复杂，但其中任何一个环节出现了错误，都可能会对整个系统的正常运行造成影响。这段不断碰壁、不断试错的经历，让我慢慢意识到，工科教育真正该培养的，不是能把理论背得滚瓜烂熟的知识存储器，而是能在各种复杂约束下把实际问题解决掉的系统构建者。课本里的理想世界和真实工程世界之间，隔着一道不小的认知鸿沟。

下面将对项目中遇到的几个核心难题以及对应的解决过程进行还原，不仅介绍相关技术细节，也进一步分析这些难题背后的技术本质以及教育层面的根源。同时会对UDIS++双阶段模型的优化思路进行梳理，从中总结一些可以迁移到其他领域的工程思维模型，最后结合项目体会，对传统工科教学模式的改革以及创新提出相关思考。

2 工程实践中的核心难题与本质剖析

项目刚进入开发阶段时，我碰到的几乎全是非算法类的技术瓶颈。这些问题不管是学术论文还是教科书里都很少专门讨论，但一个工程项目能不能真正落地，往往恰恰取决于这些问题能不能解决好。我就是在一个个解决这些问题的过程中，才慢慢对工程系统的复杂性有了比较完整的认识。

2.1 CUDA 依赖死锁

项目开发初期，CUDA 环境无法满足程序运行需求。最初尝试通过调整 CUDA 配置解决报错，但排查后发现问题主要来自代码中的设备依赖。随后进行了多种方案测试，包括更换 CUDA Toolkit 版本、调整 NVIDIA 驱动版本、建立多个 Python 虚拟环境验证 PyTorch 与 CUDA 的兼容性，以及尝试源码编译支持 CPU 的 PyTorch。然而这些方法均未解决问题，程序在模型加载阶段仍出现 “Torch not compiled with CUDA enabled” 错误，具体报错如图 1 所示。

图 1.“Torch not compiled with CUDA enabled”错误

现在回过头想，这个技术僵局之所以卡了那么久，根本原因还是我没能跳出课堂教学给我形成的思维定式。深度学习相关的课上，老师反复强调 GPU 对训练和推理有多重要，但从来没提过没有 GPU 的时候该怎么办。

时间久了，我脑子里就形成了一个固化认知。深度学习就必须用 GPU。正因为这样，我压根没往“彻底不用 CUDA”这个反向思路上去想。后来是硬着头皮去翻 PyTorch 官方文档里关于设备管理的底层实现，才慢慢摸到了问题的根子。PyTorch 的设备抽象层有个设计细节。模型权重文件会把训练时的设备信息固化下来，而 torch.load()这个方法默认会试着把模型恢复到原来的设备上。哪怕你已经设了全局环境变量禁用 CUDA，也没用。

在想明白这一点之后我就不再走修复 CUDA 那条路了，直接从根本上把程序对 CUDA 的所有依赖全部切断。这套方案最后只写了 27 行代码，就把困扰了我整整一周的 CUDA 依赖问题彻底解决了。这件事让我挺有感触的。工程问题能不能解决，很多时候不在于多高深的技术，而在于能不能跳出固有的思维，从问题本身出发去找最合适的解法。

2.2 开源库版本断裂

在解决 CUDA 问题后，程序又出现了开源库兼容问题。从本质上看，这类问题与系统碎片化以及语义化版本管理的局限有关。[5]开源库维护者为了快速迭代功能，动不动就引入不兼容的 API 变更，再加上不同库之间的依赖关系盘根错节，活脱脱一个巨大的版本迷宫。传统工科教学恰恰忽略了这个现实。讲代码的时候默认都在一个没有版本冲突的理想环境里，结果真到了真实的开源生态里，学生连最基本的版本管理和冲突解决能力都没有。

2.3 预训练模型设备绑定

经过较长时间的调整，运行环境终于搭建稳定，程序能够正常启动，模型也成功完成加载，当时认为最困难的阶段已经过去。然而在第一次上传测试图片进行拼接时，之前出现过的错误再次出现。

此时虽然已经对设备管理方式进行了修改，但程序仍然出现了 CUDA 相关错误。之后在检查 UDIS++模型代码时发现，其中直接调用.cuda()的代码并没有经过统一设备管理，因此导致之前进行的修改没有产生实际作用。

这个问题暴露出来之后，我才意识到，开源深度学习项目里其实藏着大量没写在文档里的隐性假设。这些假设在项目作者自己的开发环境里当然是成立的，但一旦换个环境，就全变成了致命的坑。除了默认有 CUDA 这一条之外，常见的隐性假设还包括操作系统默认运行于 Linux 环境、Python 版本默认设定为 3.8、内存默认大于 16GB、硬盘空间默认大于 100GB 等多种情形。这些假设文档里一个字都不会提，只有等程序真跑出错了，开发者才会反应过来原来代码是这么假设的，出错界面如图 2 所示。

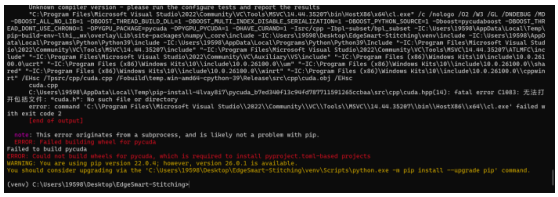

图 2.“failed building wheel for pycuda”错误

3 核心算法与模型优化的科学实践

环境和兼容性问题解决得差不多之后，我把精力转到了核心算法的优化上。UDIS++模型本身是“对齐-融合”的双阶段架构。第一阶段是 warp_model（扭曲对齐网络），负责预测两幅图像之间的单应变换和网格偏移，完成几何上的对齐，如图 3 所示。

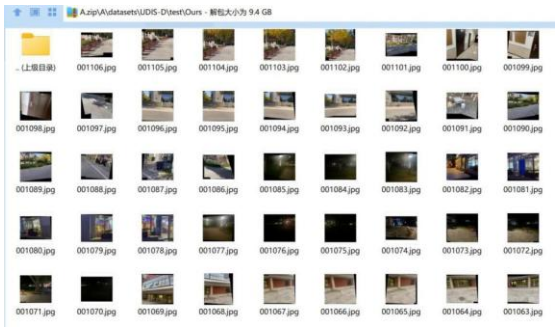

图 3.部分训练图集

第二阶段由 composition_model（融合网络）对已经对齐好的图像做像素级融合，把拼接痕迹消掉。[6]但在 CPU 边缘设备上，原始模型的推理速度远远达不到实时性要求。后来我做了比较系统的性能分析和针对性优化，才把单帧处理速度提上来。

3.1 基于性能剖面的瓶颈定位

在做优化之前需要先找准瓶颈在哪，我没有一上来就对模型做剪枝或者量化，而是先把原始代码做了性能分析。结果发现系统真正的性能瓶颈在对齐阶段的模型推理和网格计算，而不是直觉上以为的融合阶段。

3.2 对齐网络的场景化优化

对同一组固定摄像头来说，两幅图像之间的变换关系其实是恒定的，变形网格根本不需要每帧都算，算一次存起来，后面所有帧直接用就行。为了验证这个想法靠不靠谱，我专门采集了一段连续的监控视频，逐帧计算变形网格，然后统计帧与帧之间的差异。实验结果表明，只要没有外力干扰，不同帧之间的网格像素差异小到完全可以忽略不计。

这次优化完成之后，我对边缘计算形成了新的认识，最高效的优化手段未必是让算法运行得更快，而是设法让算法减少甚至完全避免执行那些根本不必要的计算。课堂上教我们的，总是怎么把通用算法做得更普适，但真到工程实践里，针对具体场景做定制化优化，往往能起到事半功倍的效果。

3.3 融合网络与前后处理的精细化优化

解决了核心瓶颈之后，我又对融合网络与前后处理做了精细化的优化，进一步挖掘系统的性能潜力。

由于 U-Net 网络要求输入尺寸需要满足一定条件，并且原始模型在处理部分图像时会出现尺寸错误，因此增加了自动填充以及裁剪功能，可以在计算前对图像尺寸进行调整，并在完成推理后恢复原尺寸。同时对部分连续算子进行了合并，从而减少重复的数据访问。在前后处理阶段也进行了相关优化，即原始代码虽然可以直接对 RGB 图像进行直方图均衡化处理，但容易造成较为明显的颜色失真。我把它改到 LAB 颜色空间的亮度通道上做 CLAHE，这样对比度提上来了，原始的颜色信息也能保住。经过上面这一系列精细化优化，系统的整体性能又上了一个台阶，同时拼接精度也没怎么掉。

4 工程实践中提炼的创新性思维模型

完成整个项目工作后，对开发过程当中遇到的问题以及处理方法进行了重新整理。相比于解决单个的技术问题，更值得总结的是在不断开展排查以及调整工作过程中形成的问题分析方式，这些经验同样可以应用于其他类似的工程任务当中。这些方法不仅在计算机视觉以及边缘计算领域具备作用，在其他工科领域当中也同样适用，可以帮助工程人员更加高效地去处理复杂问题。

4.1 反向破局思维模型

当正向路径走不通时，不妨反向思考能否直接消除问题的根源。反向破局思维的核心内涵即在于此，若解决问题的常规路径成本过高、风险过大，甚至根本无法走通，便不必在这条路上持续耗费精力，转而思考问题产生的根源，直接切断其来源，从根本上将问题消除。

4.2 隐性假设挖掘模型

项目中遇到的许多问题，并不是代码逻辑错误导致，而是程序默认条件与实际运行环境之间存在差异。比如代码默认系统跑在 GPU 上、Python 版本默认是 3.8、输入图像分辨率默认是 16 的倍数……这些假设文档里一个字都没有，只有等程序真跑崩了，我们才会反应过来原来代码是这么假设的。

经过多次开展排查工作后，逐渐总结出了发现代码隐藏条件的方法,即错误反推法。在获取了报错信息之后，反向推导代码当中默认了哪些前提条件，从而分析错误产生的缘由。例如看到 Torch not compiled with CUDA enabled 这个报错，就可以反推出代码默认 CUDA 环境是存在的。

表 1.工程隐性假设的识别与破除

核心观点维度	典型隐性假设	工程真实矛盾	解决思路
硬件依赖假设	深度学习必须使用 CUDA/GPU；PyTorch 默认 CUDA 可用	边缘设备无 GPU；CUDA 未编译导致崩溃	底层拦截 CUDA 调用；强制 CPU 推理
开源依赖冗余	项目默认含 pycuda、CUDA 相关依赖	无 CUDA Toolkit 时编译失败	清理冗余依赖；纯 CPU 最小化构建
版本兼容假设	环境/库版本固定	版本断裂；API 变更导致崩溃	锁定稳定版本；参数兼容适配
算法通用假设	算法逐帧计算；不考虑场景先验	CPU 算力不足；实时性差	网格缓存；场景不变则复用参数

4.3 约束驱动创新模型

在工程实践中，资源限制并不一定只会带来困难，有时也会促使开发者重新考虑问题，并寻找更加符合实际条件的解决方案。[9]这一道理并不复杂，资源充足时，人们往往直接选择最为省事的方案，不会去思考是否存在更为巧妙的解决途径。只有当资源不够用、被卡住的时候，才会被迫跳出常规思路，去找更高效、更巧妙的解决方案。

回到我这个项目上，CPU 算力的严格限制，其实就是逼着我放弃了“优化模型本身”这条常规路线，转而从场景的先验知识里找突破口，最后才搞出了网格缓存机制，拿到了量级的性能提升。说实话，如果当时手头有充足的 GPU 算力，我可能根本不会想到这个简单但特别高效的优化方法。

类似的情况还有不少。月球上没有空气，才逼着科学家发明了不依赖氧气的燃料电池；手机的体积一直受限，才推动工程师做出了高度集成的芯片。因此从事工程实践时，与其抱怨资源不足，不如转换视角看待问题，约束本身即是创新的契机，能否在约束条件下找到最优解，才是真正的能力体现。

4.4 分层降级容错模型

为了避免单一模块异常导致系统无法运行，我设计了三级降级方案。即当主要功能无法正常执行时，系统可以切换到其他方式以完成基本图像处理任务，从而保证程序能够继续运行。第一级是正常模式，用神经网络融合，拼接质量最好；第二级是降级模式，神经网络融合出问题的时候，自动切到传统的多波段融合，保证效果基本能用；第三级是应急模式，网格计算也失败了、缓存又没有的时候，就退化成最简单的加权平均融合，至少保证系统还能输出图像。

有了这么一套分层降级机制，系统不管在什么情况下都不会直接崩溃，鲁棒性上了一个大台阶。工程实践中有一个道理需要时刻铭记，系统整体的可靠性远比某个单一功能的完美程度更为重要。

5 认知重构与工科教育反思

把整个项目从头到尾捋一遍，这段经历带给我的成长，说比过去两年课堂上学的加起来还多，一点都不夸张。它不只是让我技术上长进了，更重要的是，它彻底改变了我对“工科学习”这件事本身的认知，也让我开始认真反思传统工科教育到底差在哪里。

5.1 个人能力的跃迁

在做项目前我只知道一些课堂中的理论知识，对于环境配置、版本管理等各种工程问题了解较少。但经过项目开发后，我逐渐认识到这些基础工作也是系统实现过程中不可缺少的部分。以前做题目总想着要找理论上的完美解，现在慢慢学会了在各种约束条件下去找够用的解。我开始理解工程领域常说的够用就好的内涵，功能、性能、成本、开发周期这几个目标之间本就需要权衡取舍，不可能全部兼顾。同时也真正体会到了系统鲁棒性到底有多重要。

5.2 对工科教学的认知补位思考与建议

完成整个项目之后，我感触最深的一点在于，当前工科教育与产业实践之间的差距，本质上并非知识点覆盖是否全面的问题，而是双方知识传授的底层范式存在根本差异。课堂上学的那一套，建立在确定性、理想化、去情境化的知识范式上；而真实的工程实践，跑的是不确定性、约束性、强情境化的问题范式。底层逻辑上的这个差别，就是为什么很多学生从课本走向实际项目的时候，都会觉得不适应、会碰壁的根本原因。[10][11]站在一个学习者的角度，我觉得工科教学至少可以从三个层面做一些认知上的补位和优化。

首先是把去情境化的知识往情境嵌入型的知识上补一补。现在的专业课程体系里，知识均以提纯后的原理形式呈现，算法只讲解核心逻辑，代码只演示功能路径，默认环境、依赖、硬件均处于理想状态。这种讲法当然效率高，能快速把理论内核传下去，但副作用是会让学生产生一种错觉，认为技术原理拿来即可直接应用。可真实工程里的知识从来都是带着具体情境的。同一段深度学习代码，在 GPU 和 CPU 环境下完全是两套运行逻辑；同一个开源库，不同版本之间可能

API 直接就断了；就算是同一种编程语言，版本迭代之后语法规则也会带来新的约束。我们在项目里花那么多精力解决 CUDA 依赖、版本冲突这些问题，说到底不是技能不够，而是从未意识到工程知识本身即包含环境与约束这一属性。在课程教学中，除了介绍技术原理，也应当说明其运行条件和可能出现的问题。环境配置、依赖管理等内容不应只依靠学生自行摸索，而可以作为工程实践的一部分进行学习。这样能更好地帮助学生理解技术在实际应用中的限制。

其次，是从良构问题求解延伸到劣构问题决策的能力训练。课堂上的习题和实验基本都属于良构问题，边界清晰、条件完备、存在唯一标准答案，主要训练的是逻辑推导和公式套用能力。但真实工程场景中遇到的绝大多数是劣构问题，目标模糊、约束繁多、信息不全，根本不存在标准答案，只有不同权衡下的可行解。以本次项目中 CUDA 适配的过程为例，这便是一个典型的劣构问题，教材上找不到对应解法，也没有标准步骤可以遵循，需要自行在修 CUDA 和禁 CUDA 两条路径之间做出判断，在模型精度与运行速度的矛盾中进行取舍，在通用优化与场景定制的策略之间做出选择。这种在信息不充分、条件模糊的情况下做分析、做权衡、做决策的能力，才是工程师真正的核心素养，但靠做标准化习题是练不出来的。所以我建议，课程体系里可以适当加一些来自真实工程的劣构问题案例，不设标准答案，鼓励学生提出不同的解法，然后论证每种方案的适用场景和代价。这种训练不是为了教某一项具体技术，而是培养学生在不确定性中自主构建解题路径的能力，这才是学生走出校园之后，面对层出不穷的新技术、新场景时真正具备的核心竞争力。

最后一点，是评价导向要从正确性慢慢往适配性上转。过去的学业评价，基本都是以结果对不对为核心标准，这种导向潜移默化地塑造了学生的对错思维，总想着寻找最优解，看到不完美的可行方案就不愿采用，也很难接受在约束下妥协这种工程逻辑。但工程实践中的评价标准本质上是适配性，不存在绝对最优的方案，只存在最贴合场景约束、综合成本最低的方案。像这样在多重约束之间找平衡的权衡思维，我认为靠单一的正确性评价是养不出来的。所以我希望实践类课程的评价能多引入几个维度，把方案的鲁棒性、兼容性、可维护性、资源效率等等这些工程指标都做一个参考，更加看重方案合理性。评价导向做这么一点微调，本质上是在引导思维方式的转变。让学生慢慢明白，工程的价值不是去追求理论上的完美，而是在给定的资源、时间、场景约束下，交付一个稳定、可靠、能用的解决方案。

5.3 新工科实践教学的改革路径

针对上面说的这些问题，结合这次项目的实践经验，我觉得新工科实践教学可以从三个方面入手改革。[12][13][14]

首先，单独建一个工程基础能力课程模块。把环境配置、依赖管理、版本控制、代码调试、单元测试、文档编写这些工程上的基础技能，整合成一门独立的必修课，放在大一或者大二上就开。通过大量动手实操，让学生在低年级就建立起工程思维、把基本功打扎实，后面学专业课、做项目的时候才不会卡在这些基础问题上。

其次，是推行问题导向的项目式教学。用真实的工程项目当载体，让学生在解决实际问题的过程中去学知识。老师从知识的灌输者变成项目的引导者和教练，来带着学生主动探索、自主学习、团队协作。在项目设计上要尽量贴近真实工程场景，把版本冲突、需求变更等等这些真实现实问题都放进去，能让学生在试错里长本事。

此外，建立一套踩坑、复盘、沉淀相结合的知识共享机制。工程实践里很多经验和教训，书本上是写不出来的，必须自己亲自踩过坑才知道。学校可以搭一个共享的工程踩坑知识库，鼓励学生把自己做项目时碰到的问题、解决过程、心得体会都记录下来，分享给后面的同学。这样一来，后来的人能少走不少弯路，记录的学生自己也锻炼了总结归纳和知识分享的能力，一举两得。

6 总结与展望

6.1 结论

做这个项目对我而言是很宝贵的经验。在解决一个个看起来不起眼但又很关键的工程问题当中，我不只是练了手上的技术，更重要的是完成了从学生到准工程师的思维转变。工科教育最终要培养的，不是会考试的学生，而是能真正解决实际问题的工程师。

6.2 展望

要培养工程实践能力，就要把系统的工程基础能力课程体系建起来，把问题导向的项目教学推下去，把开放共享的实践知识平台搭起来。这样能培养出更多既有创新精神又有实践能力的工程技术人才，给我们国家的产业升级和科技创新提供更扎实的人才支撑。[15]

参考文献

[1] 施巍松，张星洲，王一帆等。边缘计算：现状与展望 [J]. 计算机研究与发展，2019, 56 (1): 69-89.

[2] 陈鹏，胡斌，王力。边缘智能在城市安防视频监控中的应用研究 [J]. 计算机工程与应用，2022, 58 (12): 203-211.

[3] Nie L, Lin C, Liao K, et al. UDIS: Unsupervised Deep Image Stitching for Untrimmed Images [C]//Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2020: 4862-4871.

[4] 周济，赵恒，王永刚。深度学习图像拼接技术研究进展 [J]. 自动化学报，2023, 49 (5): 1015-1036.

[5] 李泽宇，吴世忠，郭涛。开源软件依赖冲突问题研究综述 [J]. 软件学报，2021, 32 (8): 2428-2452.

[6] Liao K, Lin C, Nie L, et al. Deep Image Stitching With Global and Local Mesh Alignment [J]. IEEE Transactions on Image Processing, 2021, 30: 7052-7065.

[7] 韩银和，李华伟，李晓维。边缘端深度学习模型轻量化技术研究进展 [J]. 计算机学报，2022, 45 (4): 779-808.

[8] 陈亮，张旭东，马洪兵。面向 CPU 端的实时深度学习推理优化技术 [J]. 清华大学学报(自然科学版), 2023, 63 (2): 217-226.

[9] 施炜，陈劲。约束驱动创新：资源限制下的企业创新逻辑与路径 [J]. 管理世界，2020, (11): 122-141.

[10] 林健，彭林法，易树平。新工科人才培养的核心能力与实践教学体系构建 [J]. 高等工程教育研究，2021, (1): 17-24.

[11] 顾佩华，胡文龙，林鹏。从 CDIO 到 OBE: 工程教育改革的中国路径 [J]. 高等工程教育研究，2019, (2): 1-9.

[12] 吴爱华，杨秋波，郝杰。新工科建设的核心内涵、特征与推进路径 [J]. 高等工程教育研究，2020, (1): 18-25.

[13] 张炜，王沛，刘拓。问题导向式学习在工科实践教学中的应用研究 [J]. 中国高等教育，2022, (12): 45-47.

[14] 李培根，许晓东，陈国松。工科实践教学改革：从验证性实验到探究性实践 [J]. 高等工程教育研究，2023, (2): 1-7.

[15] 教育部。教育部关于加快建设高水平本科教育全面提高人才培养能力的意见 [EB/OL]. http://www.moe.gov.cn/srcsite/A08/s7056/201810/t20181017_351887.html, 2018-10-08.