基于 AI 工程能力培养的程序设计类课程教学改革与实践

王燕** 刘靖 于美菊 崔波

内蒙古大学计算机学院，内蒙古自治区呼和浩特市 010021

摘 要 当前高校程序设计类课程的教学普遍面临三个方面的失衡：算法逻辑讲得多，工程规范讲得少；结果是否正确关注得多，调试过程怎么走关注得少；独立编码训练得多，人机协同训练得少。围绕这一现状，本文提出并在教学中检验了一套以“AI 赋能+工程导向”为核心的混合式教学改革方案，构建了“AI 提示词工程，AI 辅助调试工作流，AI 协同系统级项目实战”三级递进的能力培养路径，并设计了涵盖代码规范、内存安全、性能剖析与 AI 协作日志的四维量化评价体系。教学实践表明，借助工业级代码质量检测工具链与生成式 AI 模型，教学重心由语法纠错转向系统设计与性能优化，有效提升了学生面对复杂工程问题时的独立排查与迭代优化能力，为 AI 时代计算机程序设计类课程的改革提供了可复制、可落地的参考。

关键词 程序设计类课程；大语言模型；工程能力培养；AI 结对编程；代码质量保障

Reform and Practice in Programming Courses Based on AI Engineering Capability Cultivation

Yan WANG** Jing LIU Meiju YU Bo CUI

College of Computer Science, Inner Mongolia University, Hohhot 010021, Inner Mongolia Autonomous Region, China

Abstract—Programming courses at Chinese universities tend to show three recurring imbalances: algorithmic logic is taught far more than engineering standards, whether the result is correct gets more attention than how the debugging actually proceeds, and solo coding is practiced far more than collaborating with an AI partner. Working from this diagnosis, we designed and carried out a blended-teaching reform built around AI empowerment and an engineering orientation. The reform arranges capability-building into three progressive stages, namely AI prompt engineering, an AI-assisted debugging workflow and AI-collaborative project practice, and pairs them with a four-dimensional evaluation scheme covering coding standards, memory safety, performance profiling and logs of AI collaboration. In classroom practice, after students reviewed every AI-suggested refactoring item by item, the static code quality score rose from 62 to 89 on average; after replacing a degenerated binary search tree with an AVL tree and adding hash pre-comparison, the sorting time for 100 000 pre-sorted entries dropped from 14.20 s to 0.38 s, an improvement of about 37 times. Leveraging industrial-grade code quality inspection toolchains and generative AI models, the instructional focus shifts from syntax correction to system design and performance optimization, which enhances students’ independent troubleshooting and iterative optimization capabilities and provides a replicable, actionable reference for the reform of computer programming courses in the AI era.

Keywords—Programming courses, Large language models, Engineering capability cultivation, AI pair programming, Code quality assurance

1 引 言

软件工程的实践方式正被生成式人工智能改变，计算机专业人才培养也因此面临新的课题。GitHub Copilot、DeepSeek Coder、CodeGeeX 等大语言模型，如今已经能根据一句自然语言描述生成高质量的代码片段、自动补全上下文，甚至能解释、重构遗留代码。Gartner 在《2025 年人工智能技术成熟度曲线报告》中指出，AI 增强型软件开发已成为未来三年内最具变革潜力的技术趋势之一[1]。但在高等教育领域，尤其是被视为计算机学科“基石”的程序设计类课程中，AI 技术的渗透程度和赋能深度明显落后于产业界的实际水平。

*基金资助：内蒙古自治区研究生教育教学改革项目（面向研究生高端工程能力培养的 AI 协同教学改革与实践），内蒙古大学通识教育选修课 2.0 项目（移动互联网与 APP 应用实践），内蒙古自治区一流本科课项目（校园网组网与配置虚拟仿真实验）

**通讯作者：王燕，cswy@imu.edu.cn

程序设计类课程要培养的，是学生面对非数值计算问题时抽象数据关系、设计存储结构、分析算法效率的能力。长期以来，这类课程沿用“逻辑结构，物理存储，算法实现，复杂度分析”这条经典教学主线[2]。这套模式在训练抽象思维上确实见效，可一旦面对现代软件工程看重的代码可维护性、内存安全性、并发友好性和大规模数据处理性能，明显的“工程断层”就暴露出来了。相当一部分学生提交的 C/C++代码过不了 Valgrind 的内存泄漏检测，尽管它在 OJ 平台上已经拿到满分。多数学生遇到段错误或死锁一类的运行时异常，第一反应往往是随手改代码，或者干脆重置环境，很少有结构化的排查思路。再看 AI 这一块，大部分学生虽然都听说过通用对话式 AI，但真正能写出有效工程级提示词、解决具体技术难题的却不到一成，多数人还停留在“帮我写一个快排”这种浅层交互上[3]。

这一现状促使我们重新思考这类课程的教学目标与实施路径。以往的教学改革大多围绕翻转课堂、案例驱动或项目式学习展开[4]，却很少有研究系统探讨如何把 AI 工具当作教学生产力的一部分，嵌入课程工程能力培养的整个流程[5]。在我们看来，AI 工具的出现并不意味着可以降低对学生基础能力的要求，恰恰相反，它要求教学重心从低层次的语法记忆和模板套用，转移到更高层次的系统架构设计、异常安全保证、性能瓶颈分析与人机高效协作上[6]。为此，本文提出了一套完整的、基于 AI 工程能力培养的教学改革方案，其核心是构建一个可控的“AI，学生，教师”三元协作生态：学生借助 AI 降低重复性编码与调试的门槛，把精力放在核心算法逻辑与系统结构设计上；教师则借助 AI 设计更具挑战性的边界测试用例与反模式案例；最终，让学生完成从“能写代码”到“能写出工业级质量代码”的转变。

2 课程现状与 AI 时代的教学困境

程序设计类课程是计算机科学与技术、软件工程、人工智能等专业的核心课程[2]。当前主流的教学环境虽然已经引入了线上评测平台（OJ）和项目实践环节，但 AI 工具的普及让原有的教学痛点进一步放大，形成了三重相互交织的困境。

2.1 知识传授与工程规范的“两张皮”困境

以数据结构与算法课程为例，传统课堂讲线性表时，重点大多放在顺序表与链表插入删除操作的时间复杂度上；讲树时，重点则是递归遍历算法是否正确。这种依赖伪代码或教学简化代码的讲法，遮蔽了大量工程实现细节，学生一旦进入真实的开发场景，很快就会碰到以下几处落差。

（1）构建系统缺位：教学代码大多只是单个.cpp 文件，而工业界涉及 CMake/Makefile 构建、头文件依赖管理、动态库链接。许多学生对编译链接过程中的“未定义引用”错误一筹莫展。

（2）教材里的内存模型问题：C 语言实现往往默认使用 malloc/free，而现代 C++工程实践更看重 RAII（资源获取即初始化）机制和智能指针的使用，教学案例却很少涉及如何设计异常安全的容器类，比如在 push_back 失败时如何保证原有数据不丢失。

（3）边界条件模糊：OJ 题目大多靠预设测试用例来验证，学生很容易养成“面向用例编程”的习惯，一旦遇到测试没覆盖的极端情况，比如对空容器调用 pop、迭代器在插入后失效，普遍缺乏主动防御式的编程意识。

2.2 调试技能培养的“黑箱化”困境

调试能力往往是区分新手与熟练开发者的关键标志，但在现有的实验教学中，调试环节长期处于缺位状态[7]。

（1）实验课时本来就有限，教学时间被挤压，教师往往只能讲完算法逻辑、演示基础代码，没有余力系统教授 GDB/LLDB 命令行调试、核心转储文件分析或 AddressSanitizer 内存错误检测工具的用法。

（2）另一个麻烦是错误难以复现，学生的代码错误常常是偶发的，教师答疑时经常碰到“复现不了”的情况，教学效率也因此打了折扣。

（3）最后是 AI 带来的新问题：随着学生开始私下使用 AI 生成代码之后，调试的复杂程度反而上升了[5]。AI 生成的代码语法上通常没问题，却可能藏着极其隐蔽的逻辑漏洞，或者用了学生还没学过的高级语言特性。程序一崩溃，面对这种陌生的、由 AI 写出来的代码结构，学生往往比面对自己手写的代码还要束手无策，于是走向两个极端：要么彻底不信 AI，退回效率低下的手工编码；要么盲目信任 AI，在错误的 AI 输出里反复打转。

2.3 评价体系的“唯结果论”困境

现有的课程评价体系严重依赖 OJ 平台的自动化判题。OJ 系统在保证评分客观、提高反馈时效方面确有其价值，但评价维度偏单一[3]，主要体现在两方面。

（1）权重失衡：OJ 主要衡量的是功能层面的黑盒正确性和运行时间，代码可读性（变量命名、注释密度）、模块化程度（函数长度、执行复杂度）、内存安全性（是否存在悬空指针）以及编译过程有无警告，OJ 评价都不关注。

（2）过程也常常被忽视：学生为了通过 OJ 测试，可能会写出很糟糕的代码，例如长达 200 行的 main 函数、滥用全局变量、复制粘贴重复逻辑，屡见不鲜。这种“跑通就行”的导向，妨碍了良好工程习惯的养成，也与培养高素质软件人才的目标背道而驰。

3 工程化三级递进式培养路径设计

针对上述困境，本文提出了一条以“系统构建能力”为最终产出导向、以“AI 工具链”为效能杠杆的混合式教学路径[6]。这条路径并不是用 AI 取代传统的算法教学，而是借助 AI 填补从“算法伪代码”到“工程级代码”之间的鸿沟。具体分为三个阶段，层层递进，如图 1 所示。

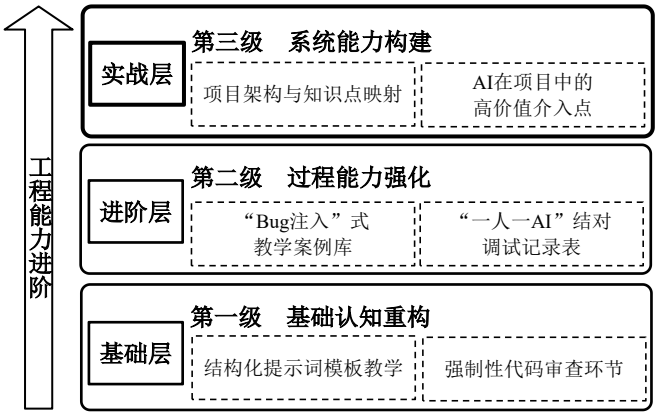


图 1 工程能力递进培育三阶段图

3.1 第一级：基础认知重构

这一阶段的教学目标不只是让学生掌握算法的基本操作，更重要的是帮学生建立起与 AI 高效协作的语言规范[5]。我们因此把“算法设计提示词工程”作为正式教学模块嵌入课程，目标是让学生完成从“会写”到“会问”的转变。

（1）结构化提示词模板教学

学生向 AI 提问的方式以前大多零散且模糊，比如“用 C++写一个链表”。这种提示词会导致 AI 生成一段缺上下文、缺内存管理、甚至不符合教学规范的代码，所以我们训练学生使用四要素提示词法。

①角色设定：明确 AI 扮演的角色，例如“具备 10 年 C++后端开发经验的架构师”。

②约束条件也要交代清楚，比如工程规范：“遵循 Google C++ Style Guide”“使用 C++17 标准”“禁止使用裸 new/delete，统一使用智能指针”“实现深拷贝构造和赋值运算符”。

③接口定义：明确对外暴露的 API，例如“提供 void push_front(const T& value)和 std::optional<T> pop_front()接口”。

④最后是错误处理策略，边界行为要写明白，例如“当链表为空时调用 pop_front，程序应 assert 或抛出特定异常，而不是返回未定义值”。

经过这样的训练，学生逐渐意识到，输入给 AI 的提示词质量，直接决定了产出代码的工程价值。

（2）强制性代码审查环节

为防止学生盲目照搬 AI 代码，我们在每次基础实验之后都加了一道人工代码审查报告环节：哪怕代码通过了所有 OJ 测试点，学生也得提交一份简短的审查报告，回答几个问题，即 AI 生成代码里有没有多余的拷贝构造开销、析构函数是否完整释放了所有节点内存、是否存在迭代器失效的潜在风险。这一举措把学生的角色从“代码搬运工”变成了“代码审计员”，逼着他们读懂每一行代码背后的逻辑与风险。

3.2 第二级：过程能力强化

这是整个教学改革的核心环节。我们不再只布置类似“二叉树遍历”的传统实验题目，而是设计了一批带有隐蔽缺陷的中型代码重构任务，并要求学生全程记录与 AI 协作排查问题的过程[4]，从而学会利用 AI 辅助调试与重构的工作流程。

（1）构建“Bug 注入”式教学案例库

我们构建了一套专门用于训练的缺陷代码库：每个案例大约 200 至 300 行代码，预置了 3 至 5 处典型的工程错误。

①案例 A（哈希表实现）：预置了迭代器在 rehash 后失效的 Bug，还有因缺少移动构造函数而导致的性能劣化。

②案例 B（图的最短路径）：预置了使用 std::priority_queue 时自定义比较器不当而引发的未定义行为。

③案例 C（多线程安全的队列）：预置了条件变量用法不当带来的死锁隐患。

（2）“一人一 AI”结对调试记录表

我们设计了标准化的《AI 辅助工程调试工作流记录表》，要求学生逐项填写，纳入实验成绩。这张表不只是作业记录，更是把学生调试思维过程显性化的工具，学生借此学会了把底层的二进制错误转化成高层次的语言描述，也学会了利用 AI 强大的知识检索能力快速理解编译器背后的抽象行为。

3.3 第三级：系统能力构建

课程末期，我们引入一个综合性的系统级项目[8]。这次改革选定的项目主题是“DictSort：基于二叉树的智能字典排序与检索原型系统”。该选题以“字典排

序”这一人人熟悉的场景作为载体，很自然地引出学生对树形结构性能与空间效率的深入探究。

（1）项目架构与知识点映射

项目要求学生做出一个命令行工具，支持 load（加载词典）、sort（排序输出）、search（单词查找）、freq（词频 Top-K）、autocomplete（前缀自动补全）等基本字典指令的子集。其中前缀自动补全模块采用 Trie 树作为核心结构，学生要实现多叉树的构建与逐字符匹配，还要探索 RadixTree 这类空间压缩技术，体会时间效率与空间消耗之间的工程权衡。词频统计模块在 BST 节点里增设计数字段，配合最小堆实现 Top-K 查询，让学生体会 BST 维护全局有序、堆维护局部最优这两种协作模式，理解“没有银弹，只有取舍”这句工程箴言。去重与快速查找模块以哈希表作为辅助索引，学生需要实践开放定址法与链地址法等冲突处理策略，并通过实验对比哈希表 O(1)查找与 BST 的 O(log n)有序遍历，认识到“数据结构的选择取决于查询模式”。作为扩展功能，同义词关联图采用有向图邻接表存储，学生需要实现 DFS/BFS 遍历与最短路径算法，完成从树到图的认知跃升。文件差异比较模块以 LCS 动态规划为核心，把编辑距离计算看作 DNA 序列比对思想在自然语言处理中的简化应用，让动态规划从理论习题真正走进工程实践。

（2）AI 在项目中的高价值介入点

在这个项目里，AI 不只是代码生成器，还充当着测试风暴制造者、性能诊断师和代码洁癖管家的角色。

①测试用例的自动化生成。学生常常只测试“加载 10 个单词然后排序输出”这种正常路径，我们因此要求学生借助 AI 生成边界测试脚本，专门攻击树结构的 corner case，覆盖空词典、重复单词、含空格与制表符或 Unicode 字符的单词、已排序输入、逆序输入以及超过 500 字符的超长单词等情形，并用系统 sort 命令交叉校验输出。效果很明显：不少学生由此发现了自己代码里的问题，包括重复单词去重逻辑缺失、已排序输入导致 BST 退化成链表、带空格单词的 trim 处理遗漏等 Bug。这种“自己攻击自己”的过程，比教师逐一布置测试用例的效率高出许多。

②性能剖析数据的 AI 解读。当学生实现了基于 BST 的基本排序之后，我们要求其导入大规模词典或自行生成万级随机词，此时程序运行明显变慢，尤其是在已排序输入的场景下。学生把 perf 报告连同自己的观察一并交给 AI，在“BST 在已排序输入下退化为链表、字符串比较函数被调用数百万次”这一现象基础上，与 AI 讨论改用 AVL 树还是 Trie 树、能否用预计算哈希优化字符串比较等具体技术选项。这种依托数据的性能优化过程，传统课堂很难覆盖到，其量化结果见 5.2 节。

③代码规范的自动化修复。项目中期检查时，教师用脚本扫描学生的代码库，再借助 AI 给出批量重构建议，重点针对字符串按值传参改为常量引用、补充空指针检查、用具名常量替换魔数等告警。此外，学生合并 AI 的修改之后还要二次确认；更重要的是，学生在审查 AI 修改的过程中，真正理解了常量引用传参避免拷贝、用空指针防御性编程的意义，这比直接拿到正确代码要深刻得多。

4 具体实施举措与四维量化评价体系重构

为了让上述三级路径真正落地，我们对教学管理流程、实验环境支撑和考核评价体系做了系统性的重构。

4.1 教学流程再造：从“讲授加练习”到“导学、协作、复盘”

传统的“理论加上机”模式被打破，重构成了新的混合式节奏[9]，具体包括三个环节。

（1）课前导学环节，布置包含 AI 提示词模板的预习任务，学生要利用 AI 生成指定数据结构的框架代码。

（2）课中协作：课堂的前 30 分钟是“AI 对抗赛”，教师展示一段故意带有隐蔽 Bug（比如浅拷贝导致崩溃）的代码，现场让学生和 AI 协作，看谁能最快定位并修复问题；教师随后点评不同学生提示词的优劣，引导全班优化提问策略，后 60 分钟则进行算法原理和复杂度的深度推导。

（3）到了课后复盘阶段，作业提交物除了.cpp 文件之外，还必须包含 AI 对话日志文件和版本历史提交记录。

4.2 四维量化评价体系的构建

为解决“唯 OJ 论”的弊端，我们构建了面向工程能力的四维评价量表[10]，如表 1 所示。总分 100 分，OJ 功能性测试只占 30%，剩下 70%都是工程能力评价项。

工程协作力的评分标准分为三档。优秀档（25 至 30 分）：对话展现出清晰的逻辑推理过程，能先自行分析现象，再提出精准的技术约束问题，并对 AI 的回复做出批判性验证，能明确指出 AI 回复中的错误或冗余。合格档（18 至 24 分）：能借助 AI 解决具体的编译或链接错误，提示词包含基本的错误信息。不合格档（低于 18 分）：对话记录为空，或者只有“帮我写代码”这类宽泛无意义的请求。

表 1 数据结构课程工程能力四维评价量表

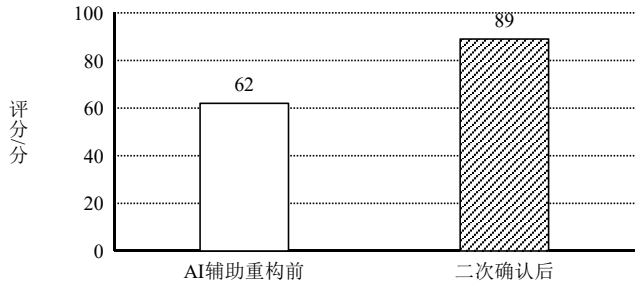
评价维度	指标描述	考核工具	权重
功能性	基础功能正确性与算法复杂度达标	Online Judge 平台	30%
鲁棒性	内存安全与异常安全（无泄漏、边界处理）	Valgrind，Clang ASan	20%
可维护性	风格统一、注释合理、模块化程度高	Clang-Tidy 报告，人工抽查	20%
工程协作力	调试流程逻辑性、提示词质量、文档完整性	教师评阅 AI 对话记录	30%

5 教学实践成效分析

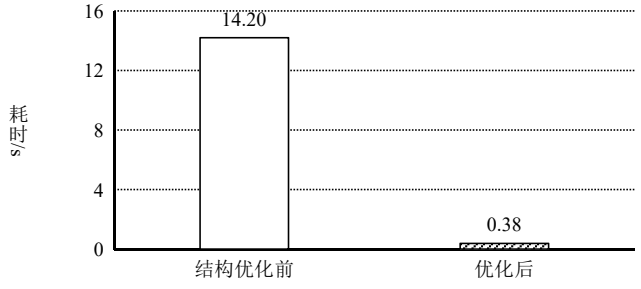
5.1 数据来源

本节数据取自算法分析与设计课程的教学班，参与学生共 85 人。数据有三个来源：一是工具链自动采集，Valgrind、Clang AddressSanitizer 与 Clang-Tidy 在每次提交时自动出具报告，用于记录内存安全与静态质量的变化；二是过程性材料，即《AI 辅助工程调试工作流记录表》与代码审查报告，按 4.2 节工程协作力的三档标准评阅；三是课程结束时的匿名问卷，回收有效问卷 81 份。受课程安排限制，本轮实践未设置平行对照班，因此下文的量化结果均为同一批学生在改革实施过程中的前后自身对照。

5.2 系统级项目的代码质量与性能改善



（a）代码静态质量评分



（b）10万已排序词条排序耗时

图 2 代码静态质量评分与系统级项目排序耗时的前后对比

在系统级项目 DictSort 的实施过程中，有两项指标取得了可直接测量的改善，如图 2 所示。在代码规范的自动化修复环节，学生合并 AI 给出的重构建议并逐条二次确认后，代码静态质量评分由平均 62 分提升至 89 分。在性能剖析环节，学生把退化为链表的 BST 替换为 AVL 树并引入哈希预比较之后，10 万条已排序词条的排序耗时由 14.20 s 降至 0.38 s，约提升 37 倍。两项数值分别由 Clang-Tidy 静态分析报告与项目性能剖析记录直接读出。

5.3 AI 协作过程质量的演进

《AI 辅助工程调试工作流记录表》在一个学期内共布置 3 次，评阅时按 4.2 节工程协作力的三档标准判定。表 2 给出三次记录表的评级分布，用于观察学生与 AI 协作质量随训练次数的变化趋势。

表 2 三次调试记录表的工程协作力评级分布（%）

记录表轮次	优秀档	合格档	不合格档
第 1 次	15%	62%	23%
第 2 次	31%	65%	4%
第 3 次	63%	37%	0

表 2 的数据说明结构化的调试记录训练确实改变了学生与 AI 交互的方式，使其从索取答案转向陈述现象、提出假设、验证结论。

5.4 学生的主观评价

课程结束时发放匿名问卷，采用 5 级李克特量表，1 表示完全不同意，5 表示完全同意。主要题项的统计结果见表 3。

表 3 课程结束问卷主要题项的评价结果

问卷题项	均值
AI 辅助使我更关注内存安全与边界条件	4.51
调试时能先形成假设再借助 AI 验证	4.73
提示词质量决定 AI 产出的工程价值	4.88
课程教授的工具链对今后开发有实用价值	4.82
AI 辅助并未削弱我独立编码的信心	4.40

从表 3 可以看出，学生对该课程的教学效果及 AI 辅助教学模式给予了高度认可，整体评价均值均处于较高水平。

6 结束语

人工智能正在改变人们获取知识的方式，在这样的背景下，程序设计类课程的教学目标不应再局限于教会学生写出正确的算法代码，而应致力于培养具备工业级质量意识、拥有高效人机协作能力的计算系统构建者。本文提出的基于 AI 工程能力培养的教学模式，通过重构“提示词工程，调试工作流，系统实战”三级递进路径，并建立多维度的过程性评价体系，较好地解决了传统教学中“懂算法却写不出工程代码”这一结构性难题。

这次改革的创新之处在于：不回避 AI 工具的使用，而是通过教学设计把 AI“关进笼子里”，用强制的提示词模板训练和过程证据留存，引导 AI 成为学生的高阶思维“陪练”，而不是简单的答案“生成器”。从第 5 节的实测数据和学生的正向反馈来看，这一模式在提升代码鲁棒性、可维护性方面的效果已经初步得到验证。

当然，教学改革没有终点。接下来的工作会聚焦两个方向：一是建设课程专属的提示词知识库，把高质量的教学互动案例沉淀成可检索的数字资产；二是进一步研究不同编程基础的学生与 AI 协作时的差异化行为模式，以便实现更精准的个性化教学干预，防止因 AI 使用不当而扩大“能力鸿沟”。我们希望通过持续迭代，摸索出一条人工智能时代计算机程序设计类课程高质量发展的可行路径。

参 考 文 献

[1] Gartner Inc. Hype Cycle for Artificial Intelligence, 2025[R]. Gartner Research, 2025.

[2] 计算机领域本科教育教学改革试点工作计划工作组. 高等学校计算机类专业人才培养战略研究报告暨核心课程体系[M]. 北京: 高等教育出版社, 2023.

[3] Stack Overflow. 2025 Developer Survey[EB/OL]. (2025-06-15)[2026-04-20]. https://survey.stackoverflow.co/2025/

[4] 方维, 袁宝库, 梁峰绮. 基于 PTA 平台的程序设计类课程教学改革实践[J]. 计算机技术与教育学报, 2022, 10(1): 97-100.

[5] 王海洋, 刘瑾, 胡静. AI 赋能 C 语言程序设计智慧课程一体化建设与实践[J]. 计算机教育, 2026(2): 91-96.

[6] 鲍鹏, 邢薇薇, 卢苇, 等. 新工科背景下人工智能实践类课程教学模式创新研究[J]. 计算机教育, 2021(6): 105-109.

[7] 凌杰, 张丽丽, 陈鸿光, 等. 生成式人工智能在计算机实验教学中的应用[J]. 计算机教育, 2025(5): 28-32.

[8] 邹世辰, 郝文宁, 程恺, 靳大尉, 相力. 专业背景课实战化教学改革探索——以“数据分析与挖掘”为例[J]. 计算机技术与教育学报，2025, 13(3): 74-79.

[9] 李家春，何克晶. 混合式教学模式下学生学习力评价和提升研究[J], 计算机技术与教育学, 2025, 13(3):11-17.

[10] 谭貌, 段斌, 周彦, 等. 面向产出落实工程教育认证标准的院系机制与实践[J]. 计算机技术与教育学报, 2023, 11(5): 16-20.