Science Innovation
2026, Vol. 14, No. 6, pp. 278–286
https://doi.org/10.11648/j.her.20261106.12

Research Article

Exploration of a Paradigm for Building LLM Knowledge Corpora for Computer Science Courses: Computer Architecture as an Example

Qin Guofeng[1], Jia Yibo[1], Xu Ting[2], Qin Minchuan[3,*], Li Qihang[1]

[1]School of Computer Science and Technology, Tongji University, Shanghai, China
[2]Shanghai Kupasi Technology Co., Ltd., Shanghai, China
[3]Independent Scholar, Shanghai, China

Abstract

Artificial intelligence is driving a new wave of educational transformation and profoundly reshaping knowledge acquisition, course teaching, and learning assessment. Computer science courses are both an important foundation for the development of artificial intelligence and a key category requiring adaptation as large language models are introduced into discipline-specific education. Compared with general-purpose question answering, computer science courses are characterized by complex knowledge hierarchies, tightly interconnected concepts, explicit computational reasoning, and substantial practical requirements. Building knowledge corpora tailored to computer science courses is therefore an important foundation for the deeper integration of artificial intelligence into course instruction. Taking the Computer Architecture course as an example, an approach is systematically discussed a method how to constructing LLM knowledge corpora for computer science courses, focusing on four main stages: establishing a chain-of-thought reasoning framework, building a course knowledge corpus, conducting domain model training, and generating questions and answers. By organizing the course knowledge system, knowledge-point reasoning trees, question-type structures, and reasoning procedures, course knowledge is transformed into training data with an explicit structure and clearly defined relationships. The paper further examines the value of such data for question answering, test-paper generation, and online tutoring, providing a practical reference for the construction of LLM knowledge corpora for computer science courses.

Keywords

Large Language Model, Knowledge Corpus, Computer Science Course, Computer Architecture, Chain-of-thought Reasoning Model

1. Research Background

To implement the national artificial intelligence development strategy, the Chinese National Data Administration, the Ministry of Education, the Ministry of Industry and Information Technology, and other departments have continued to

*Correspondence: Qin Minchuan (1253644515@qq.com)

Received: 2 October 2026; **Accepted:** 26 October 2026; **Published:** 10 November 2026

Copyright: © The Author(s), 2026. Published by Science Publishing Group. This is an **Open Access** article, distributed under the terms of the Creative Commons Attribution 4.0 License (http://creativecommons.org/licenses/by/4.0/), which permits unrestricted use, distribution and reproduction in any medium, provided the original work is properly cited.

promote the development of AI data and corpus resources. Focusing on tasks such as "building foundation models, strengthening technology, establishing platforms, and cultivating ecosystems," these efforts have advanced the systematization of corpus-data production methods, infrastructure sharing, and coordination across industry ecosystems. The integration of artificial intelligence into education and teaching has become an important direction in the digital transformation of education, with computer science courses representing a typical application domain. Computer Architecture is a core course for computer science majors. Its content spans multiple levels, including processors, memory systems, buses and input/output, and parallel processing, and is characterized by tightly interconnected knowledge, explicit reasoning processes, and substantial practical requirements [3-5]. With the concurrent development of instruction set architectures such as x86, MIPS, and RISC-V [1, 2], as well as the continued expansion of heterogeneous computing units such as CPUs, GPUs, and NPUs, the scope and applications of computer architecture continue to evolve. Against this background, this paper takes the hierarchical knowledge structure of computer architecture as its main thread and uses superscalar multiple-issue processing and vector pipelining, the structural evolution from complex instruction sets to reduced instruction sets, multi-level memory systems [6], and multi-node networked computing as its principal branches to explore the organizational relationships among the course knowledge system, the chain-of-thought reasoning process, and the training corpus [7-9].

From the perspective of course development, the key to applying artificial intelligence in education is not simply to introduce a particular model, but to reorganize knowledge that is scattered across textbooks, lecture notes, laboratory manuals, and examination questions so that models can identify, retrieve, and reason over it. A high-quality course corpus should provide disciplinary accuracy, structural completeness, interpretable reasoning, and data reusability; these are also fundamental requirements for applying large language models in computer science courses. Existing general-purpose corpora can support general language use and commonsense knowledge, but for course problems such as pipeline performance analysis, memory-hierarchy computation, instruction execution, and multiprocessor architectures, text that provides only conclusions is often insufficient to support multi-step reasoning [10-12]. Course-corpus construction therefore needs to move beyond "what the knowledge points are" to address "how knowledge points are related, how problems are solved, and how answers are verified," thereby forming a specialized knowledge representation aligned with course teaching objectives [13-15].

2. Requirements and Technical Approach for Building an LLM Knowledge Corpus

The construction of an LLM knowledge corpus can be divided into four interconnected stages: establishing a chain-of-thought reasoning framework, building a course knowledge corpus, conducting model training to develop generative reasoning capabilities, and applying the resulting model to question answering and test-paper generation. These four stages address, respectively, how knowledge is organized, how data is represented, how the model learns, and how the resulting capabilities are applied. The technical workflow is shown in Figure 1.

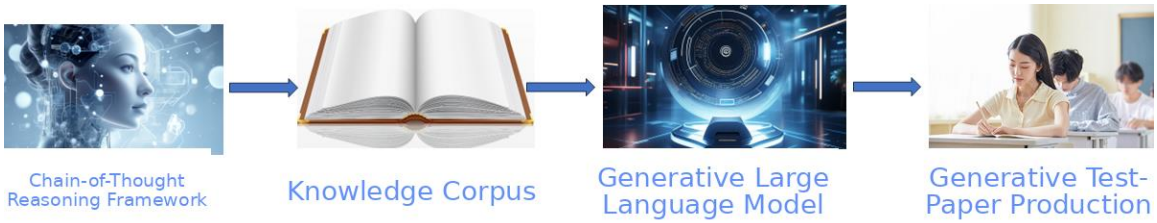


Figure 1. *Technical workflow for building an LLM knowledge corpus.*

The four stages are not independent functional modules; rather, they form a closed loop that progresses from knowledge organization to teaching applications, with application feedback informing subsequent revisions. The knowledge framework defines the coverage of the corpus, corpus quality affects model training and inference, and errors identified during model application can in turn reveal gaps in the knowledge tree, inconsistent labels, or problems in question design. Therefore, the interfaces between upstream and downstream stages should be considered together during construction to avoid inconsistent classification schemes across the knowledge tree, dataset, and application system.

The chain-of-thought reasoning framework should meet three basic requirements: (1) the disciplinary knowledge system serves as the trunk, course knowledge modules as the branches, and specific knowledge points as the nodes, forming a clearly hierarchical reasoning-tree structure; (2) reasoning paths should be concise, explicit, and traceable, allowing a

question to be traced from a specific knowledge point back to its topic and to the overall course knowledge system; and (3) knowledge points should form networked associations that cover, as far as possible, the main course content and the prerequisite, peer, and related relationships among knowledge points.

The knowledge corpus should meet three basic requirements: (1) the knowledge system should be structured, reasoning paths should be explicit, and corpus engineering should be standardized so that course knowledge can be incorporated into the dataset in a unified format; (2) question types and difficulty levels should be configured appropriately to cover different capability levels, including basic knowledge, analysis of underlying principles, quantitative computation, and comprehensive problem solving; and (3) the major knowledge points of the course should be covered as fully as possible to reduce knowledge gaps and question-type bias and to provide a relatively complete data foundation for subsequent model training, retrieval, and course applications.

Training a generative reasoning large language model should meet two basic requirements: (1) reviewed explicit reasoning chains should be used as supervisory signals to guide the model in learning the basic processes of problem analysis, knowledge retrieval, and step-by-step derivation; and (2) the course knowledge corpus should serve as the primary data input for domain adaptation using appropriate models and training methods, with iterative refinement of data and parameters to improve model stability in tasks such as course knowledge question answering and examination-question solving.

Question answering and generative test-paper production should meet three basic requirements: (1) large language models should be used to assist in generating questions, test papers, and homework explanations and to provide online learning support; (2) an examination-question evaluation and quality-assurance mechanism should be established to assess knowledge-point coverage, question difficulty, answer correctness, explanation completeness, and duplicate questions; and (3) AI-enabled educational applications should be developed in line with actual course teaching so that course resources can adapt to updates in the knowledge system and changes in teaching requirements.

The Computer Architecture knowledge corpus covers superscalar multiple-issue and vector-pipelining techniques, the structural evolution from complex instruction sets to reduced instruction sets, multi-level memory systems, and multi-node networked computing. The knowledge base is organized around three main themes: processor design, memory-system design, and bus and interface technologies. Processor design includes knowledge points such as instruction systems, instruction set architectures, pipelining techniques, and performance metrics; memory-system design includes multi-level memory hierarchies, memory performance, multi-level address translation, miss mechanisms, and handling strategies; and bus and interface technologies include interconnection networks, bus systems, input/output systems, and multiprocessor systems. The knowledge framework of the Computer Architecture corpus is shown in Figure 2.

In practical corpus production, a knowledge-point-driven approach is better suited to checking coverage, whereas a question-type-driven approach is better suited to preserving authentic teaching problems. The two approaches should be linked through unified knowledge-point codes and a data dictionary. Specifically, the knowledge tree can first be used to determine the question types and capability levels required for each node; question authors then design the questions; and reviewers finally verify whether each question genuinely covers the intended node rather than assigning superficial labels based only on terminology appearing in the question text.

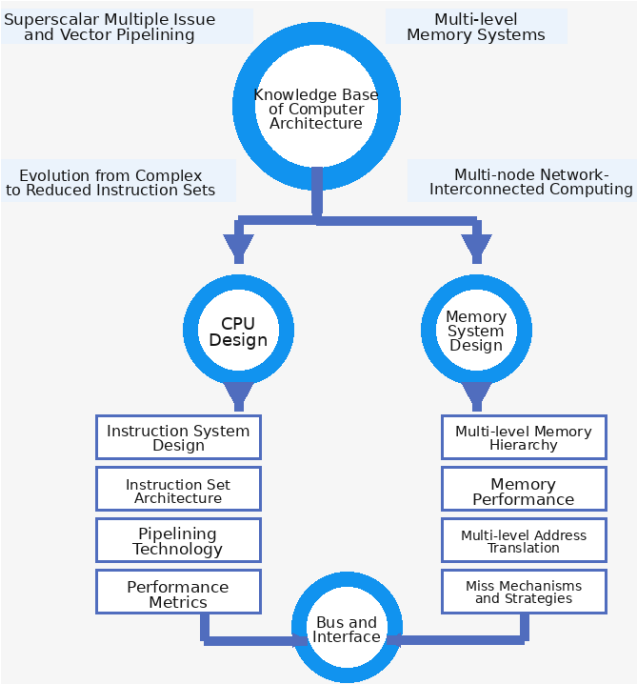


Figure 2. *Knowledge framework of the Computer Architecture corpus.*

The three themes contain both relatively independent knowledge modules and clear cross-cutting relationships. Processor design determines how instructions are executed, the memory system determines how data is accessed, and buses and interfaces support information exchange among processors, memory, and external devices. Problems involving pipeline performance computation, cache performance analysis, and multiprocessor interconnection often draw on two or even all three themes. Consequently, the knowledge framework cannot rely on a simple chapter outline alone; it must also preserve cross-topic associations to support the generation of comprehensive questions and complex reasoning data.

1) Data production, automated validation, and duplicate checking

Using the hierarchical knowledge of computer architecture

as the main thread, with superscalar multiple-issue and vector pipelining, the structural evolution from complex instruction sets to reduced instruction sets, multi-level memory systems, and multi-node networked computing as the principal branches, reasoning paths are constructed within the course knowledge base and the corpus data required for model training are designed accordingly. Data production can make use of associative and prerequisite relationships among knowledge points to create multiple question formats for the same knowledge point as well as composite questions spanning different knowledge points. After data are generated, the reasoning process, final answer, and similarity among questions should also be checked automatically, with the results submitted for manual review.

Data production can combine "knowledge-point-driven" and "question-type-driven" approaches: the former first identifies the knowledge points to be covered according to the knowledge tree and then designs questions by question type and capability level; the latter starts from existing representative questions and reverse-annotates their knowledge paths and reasoning steps. The two approaches complement each other: the former helps ensure knowledge coverage, while the latter helps preserve authentic and representative question formats used in course teaching.

2) Quality-control workflow design

Generative data include both structured data with fixed fields and explicit formats and unstructured text such as question explanations and reasoning processes. Because the two data types differ in their error patterns and checking methods, a layered and graded quality-control workflow should be designed on the basis of content-logic and data-structure checks. This study performs checks at three levels—question content, solution process, and structured data—with the aim of identifying and correcting problems before the data enter the training stage. Quality control should also distinguish between "content correctness" and "pedagogical appropriateness." Content correctness examines whether factual knowledge, calculations, and answers are valid; pedagogical appropriateness examines whether a question aligns with course objectives, whether its difficulty matches the educational level, and whether the explanation helps learners understand the problem. Even when an answer is correct, a data record should not be used directly as high-quality training data if the question conditions are incomplete, the explanation omits major steps, or the knowledge-point labels are overly broad.

The goal is to build a knowledge base and chain-of-thought reasoning corpus for the Computer Architecture course and gradually establish a mechanism for generative assessment design that covers theoretical knowledge, computational analysis, and practical laboratory content, thereby providing a data foundation for question generation, answer explanation, and learning support in course teaching.

The design of the chain-of-thought reasoning corpus for the computer architecture knowledge base is intended to build a basic knowledge framework, develop analytical reasoning capability, and strengthen the ability to solve complex problems. First, terminology definitions, short-answer questions, and true/false questions are used to construct corpus data for the AI model's basic knowledge framework. Second, multiple-choice questions, fill-in-the-blank questions, and selected analysis-and-computation questions are used to develop the model's analytical reasoning capability. Third, analysis-and-computation questions and comprehensive questions are used to develop the model's complex problem-solving capability. These components are connected sequentially and developed in stages to build a corpus with a relatively complete knowledge system and to provide a digital foundation for the AI model.

This is the first batch of corpus data. It mainly includes terminology definitions, short-answer questions, and true/false questions for building the AI model's basic knowledge framework, together with some fill-in-the-blank questions. All 2,000 original records were manually authored and compiled by the project team, and the solution process and final answer for each question were likewise produced and organized manually.

The data production workflow mainly includes drafting question stems and preparing solutions and answers; checking question stems, solutions, and answers; determining difficulty levels; defining the annotation fields for the chain-of-thought reasoning-tree vocabulary; designing the fields of the question dataset; generating the question dataset in JSON format; and checking and validating JSON-format compliance. Question difficulty is divided into three levels—low, medium, and high—based on the combined subjective judgments of the question author and the solver. A more rigorous method would be to establish an expert panel for difficulty rating and determine levels through expert scoring; however, this approach requires substantial workload and staffing and was therefore not adopted. Low-difficulty questions account for about 50% and are mainly true/false, multiple-choice, and fill-in-the-blank questions; medium-difficulty questions account for about 45% and include selected multiple-choice and fill-in-the-blank questions, explanation and short-answer questions, and computation questions; high-difficulty questions account for about 5% and consist mainly of analytical and comprehensive questions.

Quality checks are conducted during the production of questions, solutions, and answers. Project team members author the questions, and other members cross-check their correctness. Once a question is confirmed to be correct, it enters the solution-and-answer stage. After a team member completes the solution, the question stem, solution process, and answer are reviewed and discussed by other members for completeness and correctness. Each authored question undergoes one quality check, and each solution and answer undergoes another; incomplete or inaccurate solutions or answers are revised.

When the question dataset is produced in JSON format, the

fields of each record are defined according to the annotation fields of the chain-of-thought reasoning-tree vocabulary, including ID, question type (qtype), question (question), knowledge point (keypoint), answer (answer), options (choices; null for non-multiple-choice questions), explanation (explanation; records without an explanation are considered lower in quality), subject (subject), difficulty (hard_level), educational level (grade, undergraduate/graduate), language (language, zh/en), and question-bank version.

3. Knowledge-Point Reasoning-Tree Model

To assign explicit knowledge points and labels to each corpus record, a knowledge-point reasoning-tree model is constructed. For each question, the corresponding knowledge points and labels can be generated by traversing the chain-of-thought reasoning-tree model. The model is shown in Figure 3.

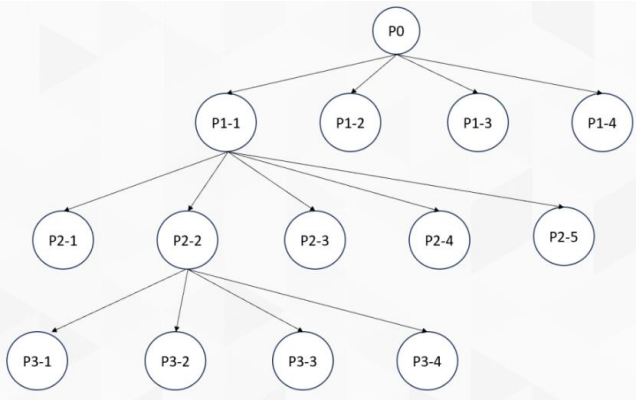


Figure 3. *Chain-of-thought reasoning-tree model.*

The chain-of-thought reasoning-tree model for Computer Architecture knowledge points is built according to the hierarchical organization of the von Neumann architecture and is structured around three aspects: processor technology, memory technology, and bus and I/O interface technology. An example of hierarchical node instantiation is shown in Table 1.

Table 1. *Examples of nodes in the chain-of-thought reasoning tree.*

Node ID	Node name	Node level	Remarks
P0	Computer architecture	Level 0	Root node
P1-1	Processor technology	Level 1	Parent node P0
P1-2	Memory technology	Level 1	Parent node P0
P1-3	Bus	Level 1	Parent node P0
P1-4	I/O	Level 1	Parent node P0
P2-1	Instruction set system	Level 2	Parent node P1-1
P2-2	Pipelining technology	Level 2	Parent node P1-1
P2-3	Instruction-level parallelism	Level 2	Parent node P1-1
P2-4	Datapath	Level 2	Parent node P1-1
P2-5	Controller	Level 2	Parent node P1-1
P3-1	Control dependence (branch hazard)	Level 3	Parent node P2-2
P3-2	Data dependence (data hazard)	Level 3	Parent node P2-2
P3-3	Pipeline performance metric: speedup	Level 3	Parent node P2-2
P3-4	Pipeline performance metric: throughput	Level 3	Parent node P2-2

4. From Metadata to Corpus Data Structures

According to the annotation fields of the chain-of-thought reasoning-tree vocabulary, each data record includes the following fields: ID, question type (qtype), question (question), knowledge point (keypoint), answer (answer), options (choices; null for non-multiple-choice questions), explanation (explanation; records without an explanation are considered lower in quality), subject (subject), difficulty (hard_level), educational level (grade, undergraduate/graduate), language (language, zh/en), question-bank version, and so on. Formulas are expressed in LaTeX format, images are referenced through URLs, and image numbers correspond to question IDs.

All corpus data were manually created and compiled, with an emphasis on uniqueness, completeness, and breadth of knowledge coverage. The readability of the JSON corpus was validated using the DeepSeek model. This batch contains 2,000 examination questions and the corresponding 2,000 corpus records, accounting for 50% of the complete 4,000-question Computer Architecture question bank. The full set of 4,000 questions comprises true/false questions (about 20%), multiple-choice questions (about 20%), fill-in-the-blank questions (about 15%), terminology-explanation and short-answer questions (about 20%), computation questions (about 20%),

and analytical and comprehensive questions (about 5%). All questions include a question stem, an answer, and a reasoning-based solution process. The following example illustrates the design and construction of corpus data according to the process and requirements described above.

Question: A five-stage pipeline (IF, ID, EX, MEM, WB) has an execution time of 10 ns for each stage. A program contains 100 instructions, where:

1) control dependence occurs with a probability of 20%, with a branch delay of 2 clock cycles;
2) data dependence occurs with a probability of 15%; forwarding resolves the dependence but requires a 1-cycle stall.

Find the actual execution time and the pipeline speedup.

Solution:

1) Ideal case: T_ideal = 5 × 10 + (100 − 1) × 10 = 1040 ns
2) Dependence overhead:

Control dependence: 100 ×20% ×2 ×10 = 400 ns

Data dependence: 100 ×15% ×1 ×10 = 150 ns

3) Actual time = 1040 + 400 + 150 = 1590 ns
4) Non-pipelined time = 100 ×5 ×10 = 5000 ns

Speedup = 5000 / 1590 ≈ 3.14

After being mapped through the chain-of-thought reasoning tree, this computation question is represented by the following corpus record.

{
"id": 12,
"uid": "cs-arch-000012",
"type": "comprehensive_question",
"language": "English",
"difficulty": "medium",
"educated_level": "undergraduate, third year",
"question": "A five-stage pipeline (IF, ID, EX, MEM, WB) has an execution time of
10 ns per stage. A program contains 100 instructions, where:\n• the probability of
control dependence is 20%, with a branch delay of 2 clock cycles\n• the probability
of data dependence is 15%, which can be resolved by forwarding but requires a
1-cycle stall\nFind the actual execution time and the pipeline speedup.",
"tag": [
"instruction pipeline",
"control dependence",
"data dependence",
"pipeline speedup",
"performance computation"
],
"analysis": "The reasoning process is as follows:\n(1) Problem identification and
knowledge-point localization: this question examines the performance computation of
a five-stage instruction pipeline under the influence of control dependence and data
dependence, involving the instruction pipeline, control dependence, data dependence,
and pipeline speedup. Given k = 5, n = 100, and a per-stage execution time
t = 10 ns, the control-dependence probability is 20% with a 2-cycle branch delay,
and the data-dependence probability is 15% with a 1-cycle stall. As implied by the
problem statement, the two types of overhead are not double counted and no other
stalls exist.\n(2) Ideal pipeline time: the first instruction must pass through all
five pipeline stages, after which one instruction completes per cycle, hence
T_ideal = k × t + (n − 1) × t = 5 × 10 + (100 − 1) × 10 = 1040 ns.\n(3) Dependence
overhead: the control-dependence overhead is 100 × 0.2 × 2 × 10 = 400 ns; the
data-dependence overhead is 100 × 0.15 × 1 × 10 = 150 ns.\n(4) Actual pipeline
time: T_pipeline = 1040 + 400 + 150 = 1590 ns.\n(5) Non-pipelined baseline:
T_non-pipelined = n × k × t = 100 × 5 × 10 = 5000 ns.\n(6) Speedup and conclusion:
the pipeline speedup is S = T_non-pipelined / T_pipeline = 5000 / 1590 ≈ 3.14.
Since 1590 ns is greater than the ideal time of 1040 ns and smaller than the
non-pipelined time of 5000 ns, the result conforms to the basic principle that a
pipeline with stalls can still improve execution efficiency.",
"figure_type": null,
"figure_text": null,
"answer": "The actual execution time is 1590 ns and the pipeline speedup is
approximately 3.14.",
"explanation": "Step 1: Identify the key concepts and extract the conditions.
This question examines performance computation and speedup calculation for a
five-stage pipeline under control dependence and data dependence. Given are the
number of pipeline stages k = 5, the clock period t = 10 ns, and the total number of
instructions n = 100; a control-dependence probability of 20% corresponds to a
2-cycle delay, and a data-dependence probability of 15% corresponds to a 1-cycle
stall; the two types of hazards do not overlap and can therefore be added
linearly.\nStep 2: Compute the ideal, hazard-free pipeline execution time. Ideally,
the first instruction must traverse all five stages to fill the pipeline, taking

k × t = 5 × 10 = 50 ns; thereafter one new instruction completes per cycle, so the
remaining n − 1 = 99 instructions take 99 × 10 = 990 ns; hence
T_ideal = 50 + 990 = 1040 ns.\nStep 3: Quantify separately the extra clock overhead
caused by the two types of hazards. For control dependence, the expected number of
instructions incurring a hazard is 100 × 20% = 20, each causing a 2-cycle bubble, so
the overhead is 20 × 2 × 10 = 400 ns, equivalent to 100 × 0.2 × 2 × 10 = 400 ns. For
data dependence, the expected number of stalled instructions is 100 × 15% = 15, each
stalling for 1 cycle, so the overhead is 15 × 1 × 10 = 150 ns, equivalent to
100 × 0.15 × 1 × 10 = 150 ns.\nStep 4: Compute the actual pipeline execution time
and the non-pipelined baseline time. Adding the ideal time and the two kinds of
overhead gives T_actual = 1040 + 400 + 150 = 1590 ns. For comparison, the
non-pipelined (fully serial) execution time is n × k × t = 100 × 5 × 10 = 5000
ns.\nStep 5: Solve for the pipeline speedup and verify its plausibility. The speedup
is S = T_non-pipelined / T_actual = 5000 / 1590 ≈ 3.14. A value greater than 1 shows
that the pipeline still delivers a clear performance gain, while a value below the
ideal upper bound of 5 reflects the significant performance loss caused by control
and data dependences; the conclusion is consistent with theoretical expectations.",
"knowledge_points": [
"instruction pipeline",
"control dependence (branch hazard)",
"data dependence (data hazard)",
"pipeline speedup",
"pipeline performance computation"
],
"source": {
"name": "Computer Architecture examination questions",
"image_urls": null
},
"created_at": "2025-11-09",
"version": "1.0",
"images": null,
"parameters": null
}

5. Test Environment and Training Validation of the Corpus Data

This study is part of a project commissioned under the Chinese Ministry of Education and National Data Administration's digital foundation program: the high-quality data corpus construction project "Construction of Chain-of-Thought Reasoning Data Corpora for a Computer Architecture Knowledge Base," whose deliverables include more than 4,000 corpus records. The relevant data were processed, trained, and validated in the computing-server environment described below. Large language models such as DeepSeek-V3 and DeepSeek-R1 were used to validate applications for answering course questions and generating test papers and answers. This section mainly describes the hardware environment used for corpus processing and model validation; the hardware configuration itself should not be interpreted as a direct measure of model-training performance.

1) Processors: AMD EPYC 7002/7003 series ×2
2) Chipset: SYSTEM OF EPYC
3) Memory: supports up to 4 TB DDR4-3200 MHz; 32 DIMM slots
4) Network: dual 10 GbE copper ports, including one RJ45 management port
5) Storage: supports 8 ×3.5"/2.5" hot-swappable drives
6) Expansion slots: 8 ×PCIe 4.0 x16
7) Power supply: 2 + 2 redundant 2200 W 80 PLUS Platinum

Table 2. Configuration of the computing server for large language models.

Item	Model	Qty
Barebone	Longjing LA4213G-8A (2+2 redundant 2200 W 80 PLUS Platinum power supply / AMD EPYC chipset / rails included)	1
CPU	AMD EPYC 3rd-gen Milan 7763 (64 cores / 128 threads / 2.45 GHz)	2
Memory	64 GB DDR4 RECC 3200 MHz	16
SSD	Intel P5520 3.84 TB U.2 enterprise SSD	1
HDD	WD 18 TB SATA 3.5" enterprise HDD	4

Item	Model	Qty
RAID	LSI 9364-8i 1 GB	1
NIC	Dual 10 GbE fiber NIC (optical modules included)	1
GPU	NVIDIA Tesla A800 80 GB data-center compute card	4
Other	Sub-server chassis (used together with the main server for storage, management, backup, and similar purposes; 2U / 550 W ×2 / 2 ×Intel Xeon 4210 (10 cores / 20 threads / 2.1 GHz) / 64 GB memory / 960 GB + 4 TB storage / dual 10 GbE fiber NIC)	2
Router	Huawei AR611E-S	1
Switch	Huawei S5735S-L24T4X-QA2	1
Monitor	27-inch 2K IPS monitor	1
Operating system	Ubuntu	1

6. Conclusion

The goal of the high-quality data corpus construction project "Construction of Chain-of-Thought Reasoning Data Corpora for a Computer Architecture Knowledge Base" is to strengthen the domain-specific knowledge representation and problem-solving capabilities of large language models in the Computer Architecture course by linking the course knowledge system, chain-of-thought reasoning framework, structured corpus, model training, and generative teaching applications. Based on the implementation of the project, this paper provides a preliminary summary of an approach that connects the structuring of course knowledge, explicit representation of reasoning paths, standardization of data production, and contextualization of teaching applications. The Computer Architecture course is highly hierarchical and engineering-oriented; using it as a case illustrates how concepts, structures, computations, and laboratory content can be incorporated into a unified knowledge-corpus construction framework. The approach summarized in this paper still requires further validation and refinement across additional computer science courses and application settings.

Abbreviations

LLM Large Language Model

Author Contributions

Qin Guofeng: Conceptualization, Resources Integration
Jia Yibo: Data Sorting and Cleaning
Xu Ting: Funding Agency, Data Acceptance Criteria
Qin Minchuan: Project Architecture, Corpus Standards
Li Qihang: Data Collection and Sorting

Conflicts of Interest

The authors all declare no conflicts of interest.

References

[1] Qin G F, Hu Y, Huang L Y, et al. Design and performance analysis on static and dynamic pipelined CPU in course experiment of computer architecture [C]. The 13th International Conference on Computer Science & Education (ICCSE 2018): 2018, pp. 1-6. https://doi.org/10.1109/ICCSE.2018.8468729

[2] Qin Guofeng, Hu Yue, Huang Linyu. Design, implementation and performance analysis of a static pipelined CPU in the Computer Architecture course experiment [J]. Education Modernization, 2018, 5(20): 183–187, 195. https://doi.org/CNKI:SUN:JYXD.0.2018-20-075.

[3] LI Xuegan. Computer architecture [M]. 5th ed. Xi'an: Xidian University Press, 2011.

[4] TAN Zhihu, QIN Leihua, HU Diqing. Practical course on principles of computer organization: from logic gates to CPU [M]. 2nd ed. Beijing: Tsinghua University Press, 2019.

[5] GAO Xiaopeng. Computer organization and implementation [M]. Beijing: Higher Education Press, 2019.

[6] QIN Guofeng, QIN Jiahao, ZOU Jianhuang, LIU Kun, HU Yue, et al. Design and implementation of a three-level memory hierarchy experiment based on Artix-7 FPGA [J]. Research and Exploration in Laboratory, 2020, 39(10): 45–49. https://doi.org/10.3969/j.issn.1006-7167.2020.10.011

[7] QIN Guofeng, HU Yue, YANG Suo. Design, implementation and performance analysis of a dynamic pipelined CPU in the Computer Architecture course experiment [M]. Tongji Education Research, Tongji University Press, 2019(1): 43–52.

[8] QIN Guofeng, WANG Lisheng, LU Youjun, et al. Laboratory manual for computer architecture [M]. Beijing: Tsinghua University Press, 2019.

[9] ZHANG Chenxi, WANG Zhiying. Computer architecture [M]. 2nd ed. Beijing: Higher Education Press, 2014.

[10] LI Yamin. Computer principles and design: Verilog HDL edition [M]. Beijing: Tsinghua University Press, 2011.

[11] HENNESSY J L, PATTERSON D A. Computer architecture: A quantitative approach [M]. 5th ed. California: Morgan Kaufmann, 2011.

[12] QIN Guofeng, DING Zhijun, WANG Lisheng, et al. Focusing on capability development and continuously advancing integrated experiments [J]. Experimental Technology and Management, 2020, 37(5): 163–167. https://doi.org/10.16791/j.cnki.sjg.2020.05.030

[13] QIN Guofeng, ZHANG Dongdong, YIN Xuefeng, WANG Lisheng, HAO Yongtao, GUO Yuchen, CHEN Yongsheng, LU Youjun. Integrating the idea of "strengthening chips and building systems" throughout the training of computer science professionals [J]. Experimental Technology and Management, 2021, 38(6): 163–167. https://doi.org/10.16791/j.cnki.sjg.2021.07.041

[14] DAVID P, ANDREW W. The RISC-V reader: An open architecture atlas [M]. St. Louis, the Bishop: Strawberry Canyon LLC, 2017.

[15] QIN Guofeng, LI Chenyang, LIN Pengpeng, WANG Lisheng, LU Youjun, GUO Yuchen. Restructuring computer systems teaching: from MIPS to RISC-V computer prototype system experiments [J]. Experimental Technology and Management, 2022, 39(5): 189–198. https://doi.org/10.16791/j.cnki.sjg.2022.05.036.