QUARCH: A Benchmark for Evaluating LLM Reasoning in Computer Architecture

Shvetank Prakash [1] **Andrew Cheng** [1] **Mark Mazumder** [1] **Arya Tschand** [1] **Varun Gohil** [2] **Jeffrey Ma** [1] **Jason Yik** [1] **Zishen Wan** [1 3] **Jessica Quaye** [1] **Elisavet Lydia Alvanaki** [4] **Avinash Kumar** [5] **Chandrashis Mazumdar** [6] **Tuhin Khare** [3] **Alexander Ingare** [1] **Ikechukwu Uchendu** [1] **Radhika Ghosal** [1] **Abhishek Tyagi** [7] **Chenyu Wang** [1] **Andrea Mattia Garavagno** [1 8 9] **Sarah Gu** [1] **Alice Guo** [1] **Grace Hur** [1] **Luca P. Carloni** [4] **Tushar Krishna** [3] **Ankita Nayak** [10] **Amir Yazdanbakhsh** [11] **Vijay Janapa Reddi** [1]

Abstract

The field of computer architecture, which bridges high-level software abstractions and low-level hardware implementations, remains absent from current large language model (LLM) evaluations. To this end, we present QUARCH (pronounced 'quark'), the first benchmark designed to facilitate the development and evaluation of LLM knowledge and reasoning capabilities specifically in computer architecture. QUARCH v1.0 provides a comprehensive collection of 2,671 expert-validated question-answer (QA) pairs covering various aspects of computer architecture, including processor design, memory systems, and interconnection networks. Our evaluation reveals that while frontier models possess domain-specific knowledge, they struggle with skills that require higher-order thinking in computer architecture. Frontier model accuracies vary widely (from 34% to 73%) on these advanced questions, highlighting persistent gaps in architectural reasoning across analysis, design, and implementation QAs. Furthermore, via fine-tuning we find that QUARCH can translate to improved performance on a realistic memory hierarchy design task, resulting in up to 1.99× more area-efficient solutions and up to 40% more viable solutions overall. By holistically assessing fundamental skills, QUARCH provides a foundation for building and measuring LLM capabilities that can accelerate innovation in computing systems. The QUARCH benchmark and leaderboard are publicly available at: https://quarch.ai/.

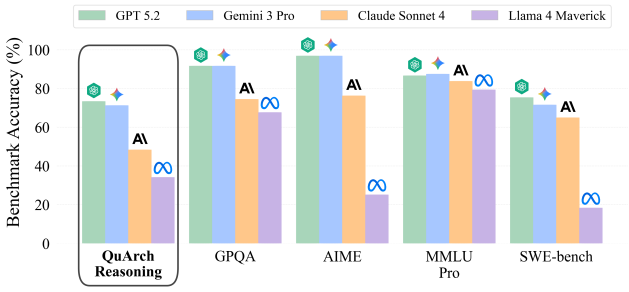


Figure 1. Reported results (Vals AI, Inc., 2025) for flagship models across QUARCH and other SOTA benchmarks to date.

1. Introduction

Benchmarks that require reasoning from large language models (LLMs) are highly sought after, as they evaluate critical thinking beyond surface-level knowledge and pattern matching. State-of-the-art (SOTA) progress is now measured by benchmarks which elicit multi-step reasoning, and models with explicit test-time deliberation (i.e., "thinking") consistently climb leaderboards. Widely adopted datasets such as GSM8K (Cobbe et al., 2021), AIME (Balunović et al., 2025), SWE-bench (Jimenez et al., 2024), GPQA (Rein et al., 2024), and MMLU-Pro (Wang et al., 2024) serve as proxies for measuring math, software engineering, and natural and physical science expertise.

Reasoning is equally central to *computer architecture*, which emphasizes evaluating trade-offs within a multi-objective optimization design space. For example, computer architects decide how to organize and balance components of systems (e.g., compute, memory, interconnects) and their power, performance, and area trade-offs. However, computer architecture remains an area without dedicated LLM benchmarks.

Existing benchmarks in computing systems target engineer-

[1]Harvard University [2]Massachusetts Institute of Technology [3]Georgia Institute of Technology [4]Columbia University [5]University of Texas at Austin [6]UC Santa Cruz [7]University of Rochester [8]University of Genoa [9]Scuola Superiore Sant'Anna [10]Qualcomm AI Research [11]Google DeepMind. Correspondence to: Shvetank Prakash <sprakash@g.harvard.edu>.

Proceedings of the 43^{rd} International Conference on Machine Learning, Seoul, South Korea. PMLR 306, 2026.

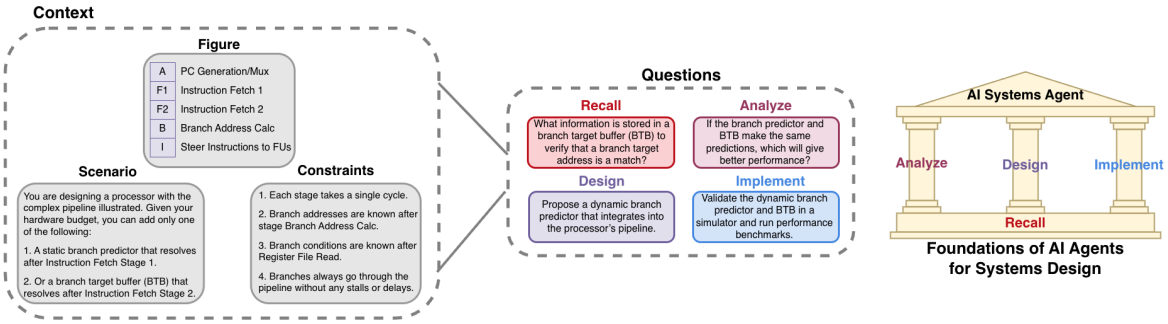


Figure 2. QUARCH QA Skills Framework. The benchmark evaluates four core competencies in systems design and computer architecture: Recall, Analyze, Design, and Implement. QAs in QUARCH contain relevant context describing the scenario, constraints, and figures when appropriate. The illustrative example shows how distinct question styles derived from the same context can probe different skills.

ing tasks for software or chip implementation such as code generation (Jimenez et al., 2024; Yang et al., 2024; OpenAI, 2024; He et al., 2025), register-transfer level (RTL) generation (Liu et al., 2023b; Pinckney et al., 2025b), system-on-chip (SoC) integration (Alvanaki et al., 2025), and chip verification (Wan et al., 2026). While these are important, they primarily evaluate whether a model can produce or manipulate programmatic artifacts, not whether it can reason about the principles that guide design decisions. Computer architecture plays a different role in the computing stack: it serves as the vital interface between software and hardware to define how these complex pieces interact, where careful orchestration of system components and their trade-offs *informs and influences implementation*. These decisions rely on conceptual understanding and analytical reasoning that is guided by application workloads and technology trends, rather than just code synthesis. Importantly, the skills required by architects to navigate these multi-objective design space problems can be systematically evaluated through a question–answering (QA) paradigm.

To this end, we introduce QUARCH: a question-answering benchmark to assess the architectural knowledge and reasoning capabilities of LLMs required in computing systems design. Figure 1 presents reported performance of frontier models across other reasoning domains in comparison to QUARCH, demonstrating that reasoning models are not yet able to solve advanced architecture questions. This gap underscores the need for focused evaluation on architectural reasoning to translate LLM progress into agentic methodologies that can accelerate innovation in computing systems.

QUARCH aims to capture the expert domain knowledge and skills that architects possess by constructing a benchmark around four foundational competencies: Recall, Analyze, Design, and Implement. Existing systems benchmarks primarily target technical implementation skills (Table 1), but all four competencies—recalling foundational principles, analyzing workloads and objectives, designing systems that balance constraints, and implementing solutions via code—are complementary and critical for effective architecture design. While we focus on computer architecture, these skills are broadly applicable to many systems tasks. QUARCH's evaluation framework enables LLM progress to be measured with fine-grained skills and compared over time.

In summary, our work makes the following contributions: ❶ **QUARCH v1.0 is the first benchmark designed to evaluate advanced computer architecture knowledge and reasoning in LLMs and is comprised of 2,671 expert-validated QAs.** 1,124 questions were curated through academic crowdsourcing and community competitions, and 1,547 questions were synthetically generated and human-verified. ❷ **To promote holistic evaluation of LLMs for systems tasks, we formalize a skills framework to systematically assess 10 frontier models on QUARCH.** Our evaluation reveals that even flagship LLMs today struggle with skills requiring higher-order thinking. Notably, QUARCH uncovers a significant performance gap between LLMs' architectural knowledge and reasoning abilities. ❸ **We conduct an in-depth analysis to offer key insights and observations on model trends and failure points**. This includes incorrect architectural assumptions made, difficulties with modeling system state, absence of architecture-semantics in code execution, and heterogeneity in LLM topic expertise. ❹ **We establish a trustworthy and scalable methodology for evaluating the correctness of free-form responses in QUARCH by comparing LLM judgments with human domain-expert verdicts across 100 QAs and 10 frontier models.** We show that LLM judgments agree with human experts at a rate of 85.48%, which is comparable to human-human grading agreement rates of 90.75% on the QUARCH benchmark. ❺ **We demonstrate knowledge transfer from QUARCH to a concrete and realistic architecture design task.** We show that fine-tuning open-source language models on QUARCH QAs can lead to improved outcomes for memory system design: Compared to base models, fine-tuned models proposed solutions that satisfied strict chip area and energy budgets in up to 40% more trials and discovered designs that were up to 1.99× more area-efficient.

Conflict of Interest Disclosure. The author A. Y. is employed by Google DeepMind, which leads the development of the Gemini and Gemma families of models, which were among the models evaluated in this paper.

2. QUARCH

2.1. Towards AI Agents for Computer Architecture

Skill Requirements. To systematically assess progress towards agentic design of computing systems, we first introduce a conceptual framework to decompose the fundamental skills that computer architects and systems engineers require. Figure 2 illustrates these skills: within a single problem scenario, we exemplify how different styles of QAs exercise different skills, from fundamental domain knowledge recall to advanced analysis, design, and implementation. Our framework is intentionally designed to align with how architects reason in practice through these coarse-grained skills, while also being informed by prior educational frameworks that employ similar competency verbs (ABET Engineering Accreditation Commission, 2024; Association for Computing Machinery, 2023).

Recall: Retrieving domain knowledge, definitions, and facts. *"What information is stored in a branch target buffer (BTB) to verify that a branch target address is a match?"* This includes the ability to identify components and roles in a diagram or specification such as standard digital logic elements. Critically, domain knowledge underpins advanced reasoning (Krieger, 2004; Duncan, 2007).

Analyze: Deducing, inferring, calculating, or interpreting data and information from a scenario to reason about workload implications and system behavior. Identifying bottlenecks and being able to explain "why" is critical for choosing trade-offs. *"If the branch predictor and BTB make the same predictions, which will give better performance?"*

Design: Proposing, inventing, or improving an architectural feature (method, component, or policy) while satisfying system requirements and constraints. It requires balancing nuanced performance, power, area, and cost trade-offs. Synthesizing a design requires iterating over architectural block diagrams and system specifications. *"Suggest a dynamic branch prediction system for this processor's pipeline."*

Implement: Translating a design into executable artifacts (e.g., code/RTL/simulation scripts). Typically, this skill is used to validate a solution via modeling or measurement. *"Implement the dynamic branch predictor and BTB in a simulator and run performance benchmarks."*

Crucially, all of these skills are significant pillars exercised in different scenarios at different times by architects and systems engineers, with domain knowledge being the foundation upon which other higher-order skills can be built. For example, without first knowing the basics of how processor execution, memory hierarchy, concurrency, parallelism, and communication work, it is difficult (if not impossible) to reason about design and performance trade-offs within a complex multi-core system.

We use this framework in Section 2.3 for benchmark characterization and Section 4.2 to analyze model capabilities across different skills.

Knowledge Breadth Requirements. Computer architecture has a multitude of specialized areas. Historical focus on microprocessor design has expanded towards many-core systems and domain-specific accelerators (Blake et al., 2009; Dally et al., 2020) due to memory and power walls (Wulf & McKee, 1995; Esmaeilzadeh et al., 2011), elevating the importance of memory systems, interconnects, and system-level methodology (Sangiovanni-Vincentelli, 2007; Carloni, 2015) to first-class concerns. *Effective architectural reasoning requires understanding relationships and interactions across these areas.* For example, a processor aggressively optimized without considering the connected memory subsystem will exhibit more performance bottlenecks than if the two were co-designed. Thus, a benchmark should capture topic breadth to properly assess architectural knowledge.

2.2. Benchmark Construction

Construction Approach. Curating a computer-architecture benchmark is particularly difficult because high-quality, openly usable sources are scarce relative to other domains (Reddi & Yazdanbakhsh, 2025) and authoring or validating benchmark entries requires substantial domain expertise to ensure technical correctness. We adopt a non-agentic QA task formulation for this domain's first benchmark because (1) it is efficient to evaluate and (2) it can assess fundamental architecture knowledge required for effective design (demonstrated in Section 4.4). To ensure realistic and challenging QAs while abstracting away iterative toolchain use, our benchmark construction process grounds questions in high-quality technical sources and incorporates both authoring and review of QAs from experts throughout QUARCH's curation. Specifically, we adopt a three-pronged strategy that combines synthetic data generation, academic exams, and expert crowdsourcing and competitions (Fig. 3). *All three strategies importantly draw from artifacts that are used to train human architects to ensure benchmark relevance to computer architecture skills and knowledge.*

Synthetic Data Generation. We collected open-source materials to curate a large corpus of computer architecture knowledge spanning technical manuals, academic publications, and comprehensive online resources. This corpus reflects a diverse and thorough survey of publicly available knowledge in the field and serves as a foundation for

Table 1. ML benchmarks & datasets across the computing stack. QUARCH broadens the scope of current benchmarks by focusing on conceptual and analytical reasoning skills required for computer architecture and systems design. Benchmarks above QUARCH target more software-oriented tasks, while those below focus on more hardware-centric, chip design tasks.

Benchmark & Dataset for Computing Systems	Focus in Computing Stack	Conceptual & Analytical QA	Design QA & Program Impl.	Multimodal Assessment	Expert Verified	Benchmark Size
SWE-bench (Jimenez et al., 2024)	Software Eng.	✗	✓	✗	✗	2294
SWE-bench Verified (OpenAI, 2024)	Software Eng.	✗	✓	✗	✓	500
SWE-Perf (He et al., 2025)	Performance Eng.	✗	✓	✗	✗	140
KernelBench (Ouyang et al., 2025)	Performance Eng.	✗	✓	✗	✗	250
CodeMMLU (Nguyen et al., 2025)	Code Reasoning	✓	✗	✗	✗	19912
CRUXEval (Gu et al., 2024)	Code Reasoning	✓	✓	✗	✗	800
QUARCH (This Work)	**Architecture**	✓	✓	✓	✓	**2671**
SLDB (Alvanaki et al., 2025)	System Design	✗	✓	✗	✓	10
CreativEval (DeLorenzo et al., 2024b)	HW Design	✗	✓	✗	✗	120
VerilogEval (Liu et al., 2023b)	RTL Generation	✗	✓	✗	✗	156
CVDP (Pinckney et al., 2025b)	RTL Generation	✓	✓	✗	✓	783
MG-Verilog (Zhang et al., 2024b)	RTL Generation	✗	✓	✗	✗	11000
EDA Corpus (Wu et al., 2024a)	EDA Tooling	✓	✓	✗	✓	1533
FIXME (Wan et al., 2026)	Verification	✗	✓	✗	✗	180
ChiPBench (Wang et al., 2026)	Layout	✗	✓	✗	✗	20

QUARCH. Using this corpus, LLMs generated cloze-style multiple-choice QAs (Rogers et al., 2023) to balance educational value with practical assessment. QAs then underwent two-stage validation: LLM-as-a-judge (Zheng et al., 2023) for initial filtering (as these cloze-style QAs naturally involve little reasoning) followed by independent review of each QA by three experts. This approach enabled the identification and removal of questions lacking definitive answers or those too narrowly scoped for meaningful assessment. Prompt details are in Appendix D.6 and D.7.

Expert Crowdsourcing & Competitions. We developed a web-based portal specifically for crowdsourcing architectural reasoning questions to target more advanced analysis, design, and implementation skills that are difficult to synthetically generate. QAs were collected via an open submission platform for individuals with technical backgrounds and time-boxed competitions. Similar to other recent benchmark curation methodologies such as Humanity's Last Exam (Phan et al., 2026), the interactive portal provided exemplary reasoning examples and real-time feedback on submitted questions to encourage participants to submit challenging questions and a solution rationale (Appendix B.2). The individual submissions and competition submissions underwent expert review to check for ambiguity and correctness before final acceptance.

Academic Exams. We additionally curated QAs from university computer-architecture exams obtained via our community crowdsourcing process and manual web scraping. A custom pipeline was developed to convert PDFs into standalone QAs. Llamaparse was first used to extract diagrams (LlamaIndex, 2025). An LLM then segmented the exam into per-question PDFs to decompose the large exam PDF and parse each QA into context, question, and solution fields. To verify parsing, QAs underwent similar two-stage validation as our synthetic data generation process that employed LLM-as-a-judge for initial filtering followed by expert review. This pipeline yielded exam-level, multimodal QAs suitable for benchmarking. Prompt details for this pipeline are in Appendix D.8, D.9, and D.10.

2.3. Benchmark Characterization

We characterize the 2,671 QA pairs of QUARCH v1.0 along architecture topics, skill focus, question format, and modality, establishing a framework for fine-grained tracking of benchmark growth over time.

Architecture Topic Diversity. QUARCH captures diverse topics in 13 core areas derived from key themes in modern computer architecture research (Figure 4). Processor architecture accounts for the largest proportion of QAs (37%), followed by memory systems (25%) and interconnection networks (8%). This distribution mirrors the field's current and historical emphasis, with niche areas containing fewer QAs. Appendix C provides example QAs that show the breadth and depth of topics covered in QUARCH. The topic distribution was estimated via two-stage classification using a text embedding model and LLM labeling (Appendix D.5).

Skills Coverage. Figure 5 characterizes QAs by the skills in Sec. 2.1 with examples for each given in Appendix C.1. We observed that many QAs naturally require multiple skills. To

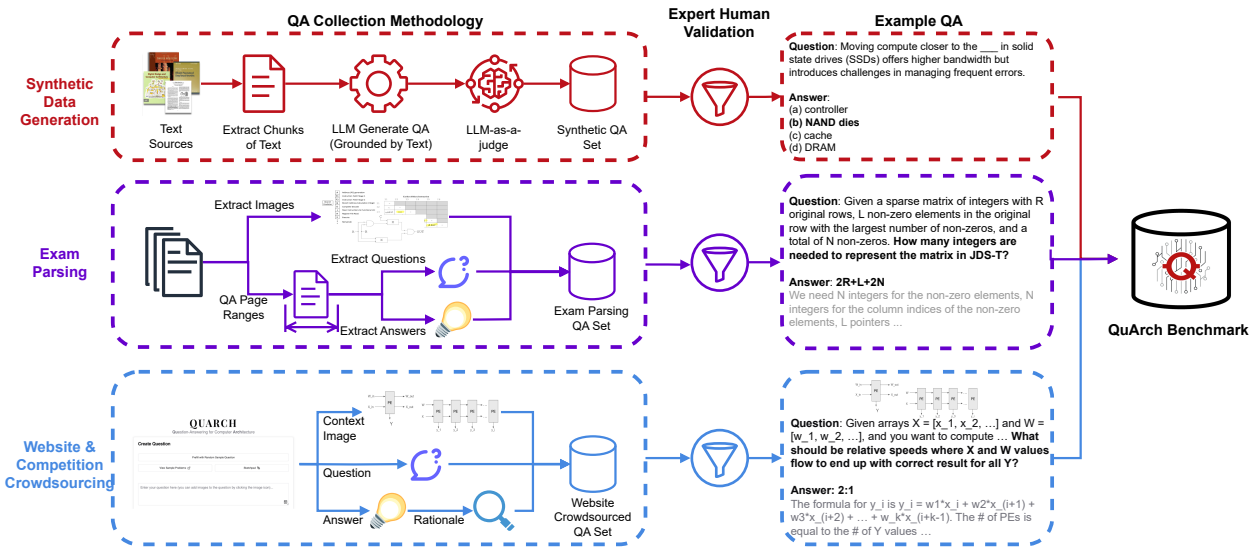


Figure 3. We construct QUARCH with a three-pronged approach including a blend of synthetic data generation, community crowdsourcing, and academic exams. All QAs are validated by a human expert to curate QUARCH's final benchmark set of 2,671 question-answer pairs.

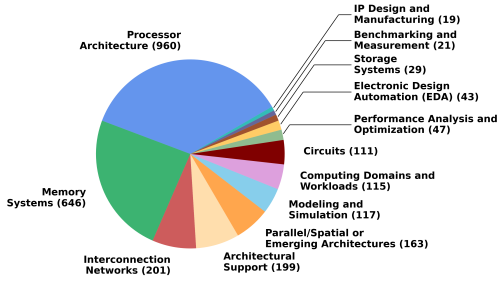


Figure 4. Distribution of topics in QUARCH.

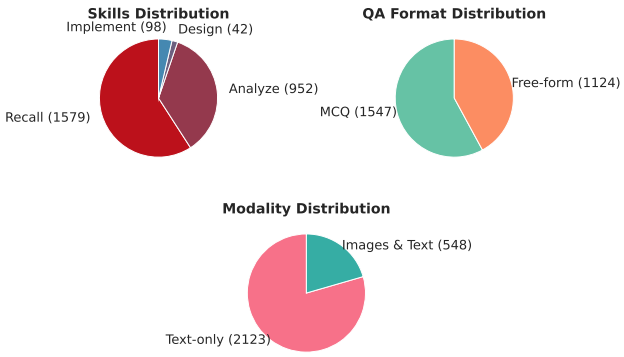


Figure 5. Breakdown of skill, format, and modality distributions in QUARCH.

better capture this nuance, all QAs were manually annotated by domain experts using a primary and optional secondary skill label when appropriate. For consistency, our analysis and characterizations use the primary skill label. We term *recall-focused* QAs QUARCH-RECALL and *higher-order skill* QAs (e.g., analyze, design, implement) QUARCH-REASONING. In particular, QUARCH-REASONING targets advanced reasoning by providing nearly 1000 analysis QAs and 140 design & implementation QAs. This delta reflects the intrinsic difficulty of authoring and validating design & implementation QAs, mirroring their natural frequency in our sources (e.g., exams typically have few design QAs relative to analysis). We importantly note that despite the limited quantity of design and implementation QAs, the graduate-level academic provenance of these questions ensures their rigor and complexity. Moreover, as shown in Section 4.3 and reflected by prior art in Table 1, small but high-quality benchmarks can meaningfully discriminate LLM reasoning capabilities and drive substantial progress.

Question Format and Input Modalities. In line with prior QA and code-reasoning benchmarks (Rein et al., 2024; Hendrycks et al., 2020; Nguyen et al., 2025), QUARCH includes 1,547 multiple-choice questions (MCQs), which are amenable to synthetic generation (Section 2.2) and have clear evaluation criteria. However, academic evaluation of domain-expert architecture knowledge is highly open-ended in structure, requiring deeper critical thinking, and thus cannot be formulated as MCQs. QUARCH therefore includes 1,124 free-response questions (FRQ), with examples in Appendix C. Furthermore, QUARCH contains both 2,123 text-only and 548 multimodal (images & text) questions. These multimodal examples assess image interpretation and reasoning capabilities on structured and spatial information, such as architecture datapath diagrams, circuit schematics, pipeline timing charts, roofline performance plots, and specification tables.

2.4. Example Questions

Figure 6 presents sample questions from QUARCH. As shown, questions in QUARCH exercise a variety of practical skills core to computer architecture. The examples include designing dataflow accelerators, configuring memory systems, and identifying hardware vulnerabilities of low-level code. Fully expanded versions of these questions, along with others, can be found in Appendix C.

2.5. Scope & Extensibility

QUARCH v1.0 is designed to evaluate the conceptual reasoning that underlies real-world architectural decision-making and design. Recent work (Mhapsekar et al., 2026; Sreedhar et al., 2025) has similarly highlighted that architectural reasoning in a QA format is a critical component of the design process and architect's workflows. Accordingly, QUARCH probes core cognitive tasks such as analyzing system performance, evaluating strategy trade-offs under constraints, and identifying bottlenecks. More broadly, QUARCH complements code and agentic benchmarks in the systems domain by isolating and measuring reasoning capabilities that influence downstream agentic performance, consistent with trends in LLM evaluation that distinguish reasoning-focused tasks from toolchain-heavy agentic evaluations (Nguyen et al., 2025; Dinella et al., 2024; Gu et al., 2024). Critically, QUARCH is extensible by design: our public submission portal (Appendix B.2) continues to receive new submissions from academic and industry contributors, and our skills framework (Fig. 2, Sec. 2.1) provides a foundation for extending QUARCH toward future evaluations that incorporate executable artifacts for design and implementation tasks, enabling the benchmark to grow in scope and difficulty alongside advancing frontier model capabilities.

3. Experimental Setup

Models. We evaluate 10 frontier models from Anthropic, DeepSeek, Google, Meta, Mistral, and OpenAI on QUARCH v1.0 across the four skills (Recall, Analyze, Design, Implement) presented in Section 2.1. Evaluation results for an additional 20 models are reported in Appendix B.5.

Evaluation grading. All models are evaluated in a zero-shot setting. Full evaluation prompts are provided in Appendix D. For MCQ-style responses, models must conclude with the correct choice of A, B, C, or D. For FRQ-style questions, we employ LLM-as-a-judge (Zheng et al., 2023), tasking an external model to assess the correctness of an answer with respect to the ground truth. We further motivate and rigorously validate our use of LLM-as-a-judge in Section 4.5.

Metrics. For both MCQ- and FRQ-style questions, we report model performance using "per-generation accuracy", the percentage of correct answers received out of n total responses, to provide a better estimate of the *pass@k*=1 metric (Pinckney et al., 2025a) under stochastic generation. In all evaluations, we generate $n = 3$ samples per question.

Table 2. Frontier model performance on QUARCH. Reported values are the per-generation accuracy across 3 generations. All models struggle much more on QUARCH-REASONING compared to QUARCH-RECALL. We highlight the **first**, **second**, and **third** best performing models.

Model	QUARCH		
	Recall	Reasoning	Δ
Multimodal Models			
GPT-5.2	**88.4**	**73.2**	**-15.2**
GPT-5 (Non-Reasoning)	86.4	48.4	-38.1
GEMINI-3-PRO	**90.9**	**71.0**	**-19.9**
GEMINI-3-FLASH	**90.1**	**71.9**	**-18.2**
CLAUDE-SONNET-4	85.3	48.1	-37.2
CLAUDE-3.7-SONNET-THINKING	85.7	51.8	-33.9
LLAMA-4-MAVERICK	85.2	33.7	-51.5
MISTRAL-MEDIUM-3.1	84.2	33.8	-50.4
Text-Only Models			
GPT-OSS-120B	84.5	63.6	-20.8
DEEPSEEK-R1	86.9	55.4	-31.5

4. Evaluation & Analysis

4.1. Model Performance

Table 2 reports headline accuracy of 11 frontier models on QUARCH. As defined in Section 2.3, QUARCH-REASONING covers higher-order skills in our framework, while QUARCH-RECALL captures domain-knowledge retrieval (rather than reasoning). QUARCH-RECALL performance is consistently strong across all frontier models. Including a recall split is useful to establish a baseline in a field that lacks a dedicated benchmark: the split distinguishes "don't know" from "can't reason," and informs whether fundamental domain knowledge is present in models. Unlike frontier models, current small language models (SLMs) exhibit gaps on recall performance (Appendix B.3). This is significant as SLMs are becoming increasingly important for agentic AI (Belcak et al., 2025). Overall, the reasoning variant of GPT-5 leads on QUARCH today with Gemini models forming the next tier. We note that GPT-OSS-120B and DEEPSEEK-R1 are evaluated only on text (no images), so their scores reflect text-only capability. We focus the rest of our analysis on QUARCH-REASONING because it offers the most headroom for today's frontier models to improve.

4.2. Skill Performance Trends

Table 3 provides fine-grained skill-wise performance across models, with key trends shared below:

(1) Recall is Mastered, Higher-Order Skills are Not. Fron-

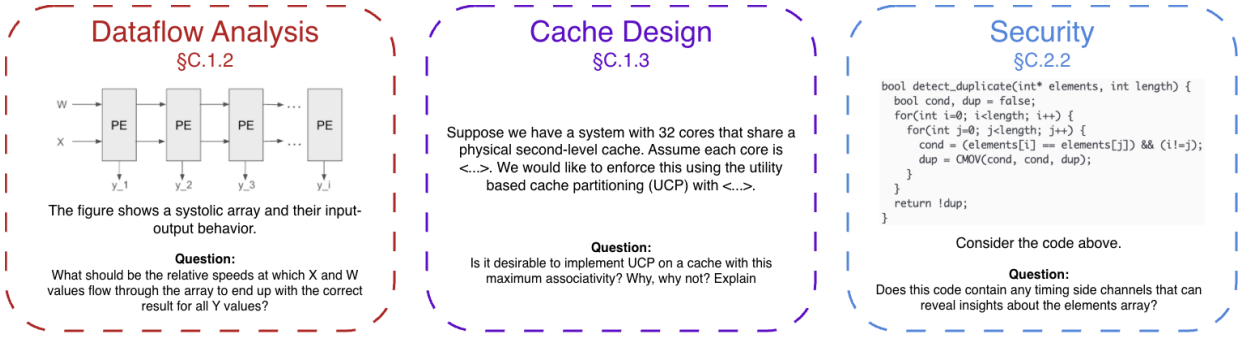


Figure 6. Example Questions from QUARCH. Full question details in Appendix C.

tier models have largely mastered recall, but fall short on advanced skills. Recall accuracy ranges between 84%-91%, suggesting architectural knowledge is present. However, analyze, design, and implement skills are lower than recall by 31%, 31%, and 38% on average respectively. Notably, multiple models with strong recall accuracy fall below 35% on other skills (e.g., LLAMA-4-MAVERICK, MISTRAL-MEDIUM-3.1). In particular, implement skills exhibit the widest performance gaps, ranging between 24-72%. This suggests the design QAs, despite comprising a small proportion of the dataset, provide a strong discriminative signal that correlates with performance across the full QUARCH benchmark.

(2) Reasoning Matters for Advanced Skills. Results from (1) indicate that translating domain knowledge into advanced skills will require targeted training and test-time deliberation mechanisms. Comparison of GPT-5.2 with a non-reasoning variant supports this. GPT-5.2 outperforms GPT-5 (Non-Reasoning) by 24%, 22%, and 33% on analyze, design, and implement QAs respectively, compared to a much smaller 2% lift on recall QAs.

(3) Variations in Competencies Across Models. Our skill framework exposes model-specific strengths and failures that a single aggregate score hides. For instance, 23 of the 30 models evaluated (Appendix B.3) score lower on implement QAs than on design QAs, underscoring the need to target all of these higher-order skills to holistically assess architecture competency.

4.3. Key Insights & Observations

Results on QUARCH illustrate a clear gap in model capabilities. Based on extensive grading performed by experts (Section 4.5), we synthesize key insights into specific failure modes observed.

(1) Struggles with architecture-semantics of code execution. Architectural semantics of code execution refers to the deep understanding of how high-level code interacts with the underlying hardware architecture (e.g., memory access patterns, instruction scheduling, etc.) (Tschand et al., 2025). Our analysis reveals that LLMs struggle with these nuanced aspects of code execution, failing to accurately predict or analyze the architectural implications of code snippets (Appendix C.2).

Table 3. Per-generation accuracy (%) by QUARCH Skill. Best performing models highlighted **first**, **second**, and **third**.

Model		QUARCH-REASONING		
	Recall	Analyze	Design	Implement
Multimodal Models				
GPT-5.2	**88.4**	**73.0**	**79.4**	**72.4**
GPT-5 (Non-Reasoning)	86.4	48.9	57.1	39.5
GEMINI-3-PRO	**90.9**	**71.4**	**73.8**	**65.6**
GEMINI-3-FLASH	**90.1**	**72.8**	**66.7**	**66.3**
CLAUDE-SONNET-4	85.3	48.8	50.0	41.2
CLAUDE-3.7-SONNET-THINKING	85.7	52.8	44.4	45.2
LLAMA-4-MAVERICK	85.2	34.6	26.2	28.0
MISTRAL-MEDIUM-3.1	84.2	34.6	38.1	24.5
Text-only Models				
GPT-OSS-120B	84.5	63.9	66.7	58.1
DEEPSEEK-R1	86.9	56.6	47.8	44.4

(2) Assuming unconventional architectural properties. In computer architecture, decades of practice have cemented certain system designs, such as byte-addressable memory, as de facto defaults unless otherwise specified. However, our analysis exposes a misalignment: when prompts fail to state conventions explicitly, we observe LLMs defaulting to unconventional choices, such as word-level addressing (Appendix C.3). Models are able to succeed when provided with the default conventions explicitly, highlighting that practitioners leveraging LLMs in this domain must identify their implicit assumptions to guide the model effectively.

(3) Modeling and tracking system state. Building an intuition and mental model of how system components interact and the implications of their interactions is central to computer architecture. In general-domain QA, this corresponds to situational world modeling (Rogers et al., 2023): instantiating entities, tracking their locations and states, and inferring temporal and causal relations to answer queries about an evolving scenario. We find that frontier models often fail to maintain consistent system state and thus misunderstand how local actions cascade into system-level effects on latency, throughput, and correctness (Appendix C.4).

Table 4. Percentage of trials in which at least one proposed solution satisfied both area and energy design specifications.

	Baseline	Fine-tuned
GEMMA-3-27B-IT	6 / 20 (30%)	14 / 20 (70%)
LLAMA-3.3-70B-INSTRUCT	11 / 20 (55%)	19 / 20 (95%)

Table 5. Confusion matrix comparing LLM-as-a-Judge with domain-expert human grading on FRQs.

	Human Correct	Human Incorrect
LLM Judge Correct	453	47
LLM Judge Incorrect	51	374

(4) Variations in domain expertise. Our analysis reveals that LLMs develop specialized expertise across different domains. For instance, within "Implement"-Style questions, GPT-5.2 performs well on Emerging Architectures and struggles on IP Design, while GEMINI-3-FLASH exhibits the opposite behavior. While overall, GPT-5.2 performed noticeably better on "Implement"-Style questions (6% improvement), these domain expertise differences resulted in GEMINI-3-FLASH performing ∼50% better on IP Design implement questions. Model capabilities are thus more nuanced than the aggregate scores of Table 3. These findings provide the opportunity to create multi-model systems that combine the domain strengths of multiple LLMs rather than relying on a single "best" model. Spider plots visualizing these per-topic variations for frontier models are shown in Appendix B.4.

(5) Sensitivity to QA modality. In computer architecture, visuals such as pipeline diagrams, cache hierarchies, and system interconnects convey structural relationships and spatial information that cannot be adequately captured through text descriptions alone (Chang et al., 2025). Multimodal models perform on average 5% worse on questions with images than on text-only questions (Appendix Table 7). This gap indicates that frontier models struggle with interpreting and reasoning about diagrams, schematics, and tables (see Appendix C.5 for failure examples).

4.4. Knowledge Transfer Case Study: Applying QUARCH-REASONING QAs to Memory Design

This section investigates the following question: *Can a model's ability to solve questions in QuArch translate to improved success on a concrete architectural design task?* To answer this, we conduct a case study that asks models to design the memory hierarchy of an ML hardware accelerator for matrix multiplication, an extensively studied problem in computer architecture (Sze et al., 2017). Critically, effective memory hierarchy optimization demands reasoning grounded in core architectural principles to understand the complex interactions between design knobs that impact area–energy trade-offs. A second case study focused on memory controller design is in Appendix B.7.

Task Formulation. Models must jointly propose hardware and software modifications (i.e., cache hierarchy organization, dataflow ordering, tiling strategies, memory layout) with the objective of reducing chip area relative to a baseline design while satisfying strict energy constraints. Proposed designs are evaluated using established architectural simulation and area/energy estimation tools (Parashar et al., 2019; Wu et al., 2019). See Appendix B.6 for additional details.

Training Methodology. We select two open-source models for training: GEMMA-3-27B-IT and LLAMA-3.3-70B-INSTRUCT. We distill answers and explanations on a subset of QUARCH (45 text-only FRQs related to memory subsystems and matrix multiplication) into these models via supervised fine-tuning (SFT) (Muennighoff et al., 2025). In total, we fine-tuned on 541 LLM responses to these 45 questions and 135 GPT-5.2 explanations on how to solve each question, for three epochs. See Appendix B.6 for hyperparameters.

Results. For both the base and fine-tuned models, we perform 20 independent trials of this task, where one trial consisted of 10 iterative turns between the model and the design simulator. Table 4 reports the percentage of trials in which the model successfully produced a solution that met the chip area and energy budget provided for this design. We observe that fine-tuning models on QUARCH leads to a significant increase in the percentage of proposed designs that met chip area and energy budgets compared to the two baseline models. With an additional epoch of fine-tuning on the same dataset, the fine-tuned Gemma model is able to discover a design requiring much less chip area at $3608.89\mu m^2$, a $1.86\times$ improvement over its base model counterpart (Appendix Table 9). Similarly, the Llama model finds a solution requiring only $3637.89\mu m^2$ ($1.99\times$ improvement over its base model), albeit at the trade-off of lower success rate across trials. *This case study exemplifies how the knowledge required to answer QuArch QAs can be applied to improve on tasks asked of computer architects.*

4.5. LLM-as-a-Judge Analysis

Motivation. Semantically equivalent and correct solutions to the same FRQ can differ in phrasing, as shown in Appendix C.6. Since full manual grading by domain experts is intractable, we employ LLM-as-a-judge for QUARCH.

Human Validation. While LLM-as-a-judge has gained popularity for evaluating FRQ-style questions (Lee et al., 2024; Zhou et al., 2023; Mañas et al., 2024; Pinckney et al., 2025b), the approach is still relatively new. We therefore validate the fidelity of LLM-as-a-Judge by measuring agreement

rates between human expert and LLM judge verdicts on the correctness of generated FRQ responses. We randomly sampled 100 (8.9%) freeform QAs in QUARCH, and generated one response each from 10 models[1]. We tasked a cohort of 11 domain experts in computer architecture and hardware design to manually grade the resultant 925 responses[2] as CORRECT, PARTIALLY-CORRECT, or INCORRECT. For analysis purposes, PARTIALLY-CORRECT is recategorized as INCORRECT. Each question is graded independently by up to 3 experts and the majority consensus is taken. All LLM judge evaluations in Sec. 4.1 and Appendix B.5 are likewise performed 3x and the majority vote taken.

LLM judges agree with human experts. In Table 5, we compare CLAUDE-3.7-SONNET-THINKING as a judge LLM against expert humans and observe an agreement rate of 89.41% with single LLM judgments against human consensus (see Appendix B.9 for details). We compare this agreement rate with the rate that expert humans disagree on verdicts. 84 of the 925 responses required a third expert to adjudicate between a correct and incorrect vote, corresponding to a human-to-human agreement rate of 90.92%. Since this inter-human agreement rate is comparable to the rate that our LLM-as-a-Judge agrees with human consensus, we argue LLM-as-a-Judge is eminently suitable for scalable and informative benchmarking of model performance on QUARCH. Additional experiments on human expert grading difficulty, alternative judge LLMs, and consensus rates are included in Appendix B.9.

5. Related Work

Software. Function-level code efficiency benchmarks (Du et al., 2024; Huang et al., 2024; Shypula et al., 2023; Waghjale et al., 2024) and domain-focused performance tasks (Press et al., 2026; Ouyang et al., 2025) evaluate correctness-preserving edits and runtime gains at the function or kernel level. Repository-scale SWE benchmarks and agentic toolchains (Jimenez et al., 2024; Yang et al., 2024; Wang et al., 2025) test long-horizon code manipulation and integration. Recent QA code understanding benchmarks (Gu et al., 2024; Nguyen et al., 2025; Li et al., 2024; Dinella et al., 2024) target control/data-flow semantics, behavioral equivalence, and code review comprehension. Unlike QUARCH, which primarily targets pre-implementation, system, and architectural judgment, these works focus on code artifacts, assessing code semantics rather than system-level design.

Hardware. Domain-specific foundation models for chip design (Liu et al., 2023a), electronic design automation (EDA) tool interaction (Wu et al., 2024a;b), RTL generation (Chang et al., 2025; Thakur et al., 2024; Liu et al., 2024; Blocklove et al., 2023), design optimization (Chang et al., 2023; Pei et al., 2024; DeLorenzo et al., 2024a), and security-oriented tasks (bug repair and assertions) (Tsai et al., 2024; Yao et al., 2025; Fu et al., 2023; Pearce et al., 2023; Nair et al., 2023; Meng et al., 2024; Mali et al., 2024) emphasize producing or improving implementation artifacts and driving tools. In contrast, QUARCH isolates the reasoning that guides implementation (e.g., architectural trade-offs) rather than their ability to generate HDL/RTL or steer EDA flows.

QA benchmarks. General-purpose and domain QA datasets (Rajpurkar et al., 2016; Trischler et al., 2017; Clark et al., 2019; Hendrycks et al., 2020; Rein et al., 2024; Huber et al., 2022; Jin et al., 2019; Cobbe et al., 2021; Zhong et al., 2020) have been instrumental for advancing and measuring LLMs (Rogers et al., 2023). QUARCH targets advancing computer architecture specifically, with expert-verified items and skill-wise evaluation capabilities not covered by existing QA benchmarks.

6. Conclusion

We introduce QUARCH v1.0, the first benchmark to directly assess computer architecture knowledge and reasoning in LLMs across four complementary skills: Recall, Analyze, Design, and Implement. Evaluating ten frontier models on 2,671 expert-validated QAs, we find consistently strong recall across models but reveal a pronounced gap in higher-order abilities that demand architectural reasoning. By providing insights into failure modes and enabling systematic tracking, QUARCH lays the groundwork for accelerating AI progress in computer architecture and, more broadly, in reasoning-centric skills for systems design.

Acknowledgments

We thank the anonymous reviewers for their thoughtful feedback and suggestions, which helped improve the quality and clarity of this manuscript. We also extend our gratitude to Derek Lockhart, Cliff Young, and James Laudon for their valuable feedback on the paper, as well as the extended team at Google DeepMind for supporting this research direction. We especially thank Kai Kleinbard for developing the QUARCH website and managing project infrastructure throughout the development of this work.

Finally, we would like to acknowledge and sincerely thank the many students and contributors who played an important role in the development of QUARCH. This project was made possible through a large collaborative community effort spanning dataset creation, question verification, infrastructure development, and evaluation. The list that follows includes all contributors, ordered alphabetically by first

[1] Model list in Appendix B.9.

[2] Non-multimodal models cannot generate responses for the multimodal proportion of sampled questions.

name, who had participated in the QUARCH v1.0 project at the time of submission (January 28, 2026): Aarush Gupta, Abhiram Ghanta, Adarsh Sriuma, Aditya Bhaskar, Aditya Borse, Aditya Mavle, Ajay Joshi, Akash Bommidi, Alexander Snapp, Andrej Vrtanoski, Andrew Peng, Ankith Thalanki, Ansh Bhatti, Anuj Bhatt, Anurag Yadav, Arkaprava Basu, Arkapravo Ghosh, Aryan Gupta, Avani H., Avi Kapur Srinivasan, Ayushi Rajpoot, Bujji Selagamsetty, Cheng-Jhih Shih, Chiranjeevi Chimmili, Daniel Terrell, Dhruv Raj Bangad, Divya Mahajan, Eli Corley, Ellen Suh, Eugene Chu, Euijun Chung, Fnu Navneet, Gaurang Upasani, Gowsika Dharmaraj, Han Cho, Hanran Wu, Haomei Liu, Hema Chandra Kolisetty, Himanshi Gupta, Hongzheng Chen, Hsueh-Yuan Chou, Hunter Lee, Ian Wong, Isaac Khor, Ishita Vohra, Ismael Youssef, Jackie Mac Hale, Jae Hyung Ju, Jagadheesvaran Tirupathi Subburayan, James Xu, Jarvis Jia, Jeeho Ryoo, Jenny Huang, Jessica Hernandez, Jiajie Qian, Jiayi Qian, Jingtian Dang, Jinhyeok Park, Jogesh Kumar, Joseph Ferraro, Joshua San Miguel, Jun Liang Ho, Kai Kleinbard, Kaiyi Hu, Kalp Vyas, Kevin Sui, Krishil Gandhi, Laith Shamieh, Lexington Whalen, Lizy K John, Logashree Venkatasubramanian, Marian Verhelst, Mayur Peshve, Miaoyan Zhou, Minseung Jung, Mohamed Ghanem, Mohnish Pai, Muhammad Haseeb, Nathan Duggal, Nathan Zhong, Nicolás Majorel Padilla, Nishant Gadde, Noah Bruckner, Onur Mutlu, Panya Bhinder, Philip Ndikum, Pin-Jun Chen, Po-Han Porras Huang, Pooria Taheri, Prabhav Gupta, Pramath Balisavira, Pranaav Milaganur Mohan, Pranay Jaggi, Pratham Nandy, Pravesh Sanjay Jamgade, Priya Panda, Puneet Bansal, Rahul Raj, Ramil Agliamzanov, Renan Silva, Ribhu Das Purkayastha, Royce Arockiasamy, Saketh Patel, Samuel Xu, Sanjay Patnala, Santosh Pandey, Saurabh Singh, Seungjae Jason Lee, Shaunak Ghatpande, Shehab Naga, Shengjie Lin, Shiv Prakash, Shreya Chivilkar, Shreyas Grampurohit, Siddharth Joshi, Sidney Wright, Soham Chausalkar, Soham Rattan, Sri Siddarth Chakaravarthy Prakash, Srinath Suresh Kumar, Srinidhi Subramaniam Pasupathy, Steve Tang, Suyash Dandekar, Tianwei Jiang, Tianyou Zhao, Twisha Shah, Varun Komperla, Vimalan Krishnan Manivannan, Wells Lu, Yashas Ambati, Yasmein Khalil, Ye Chen, Youssef Jaafar, Yuehan Zhang, Yuhao Zhu, Yujie Li, Yuming Chang, Yunchuan Zhang, Yunnuo Zhang, Yuxiang Wei, Zebin Guo, Zekai Wang, Zekun Li, Zhewen Pan, Ziyao Yin.

This work was supported by NSF Grant CCF-2324862 and POSE-2346173. This work was also supported in part by the U.S. DOE DeCoDe Project No. 84245 at PNNL and by the Columbia Center of AI Technology (CAIT). The authors were additionally supported by the NSF Graduate Research Fellowship Program (GRFP). We also thank Together.AI and Amazon Research Awards (ARA) for providing research credits that supported the fine-tuning and model evaluations conducted in this work, respectively.

Impact Statement

This paper presents work whose goal is to advance the field of machine learning. There are many potential societal consequences of our work, none of which we feel must be specifically highlighted here. Our statement on reproducibility is provided in Appendix A.

References

ABET Engineering Accreditation Commission. Criteria for accrediting engineering programs, 2025–2026. https://www.abet.org/wp-content/uploads/2024/11/2025-2026_EAC_Criteria.pdf, 2024.

Alvanaki, E. L., Lee, K., and Carloni, L. P. SLDB: An End-To-End Heterogeneous System-on-Chip Benchmark Suite for LLM-Aided Design. In *2025 IEEE International Conference on LLM-Aided Design (ICLAD)*, pp. 227–234, June 2025. doi: 10.1109/ICLAD65226.2025.00016.

Association for Computing Machinery. Bloom's taxonomy for computing. https://cecc.acm.org/assessment/blooms-for-computing, 2023.

Balunović, M., Dekoninck, J., Petrov, I., Jovanović, N., and Vechev, M. Matharena: Evaluating llms on uncontaminated math competitions. https://matharena.ai/, 2025. SRI Lab, ETH Zurich.

Belcak, P., Heinrich, G., Diao, S., Fu, Y., Dong, X., Muralidharan, S., Lin, Y. C., and Molchanov, P. Small Language Models are the Future of Agentic AI, September 2025.

Blake, G., Dreslinski, R. G., and Mudge, T. A survey of multicore processors. *IEEE Signal Processing Magazine*, 26(6):26–37, 2009. doi: 10.1109/MSP.2009.934110.

Blocklove, J., Garg, S., Karri, R., and Pearce, H. Chip-chat: Challenges and opportunities in conversational hardware design. In *2023 ACM/IEEE 5th Workshop on Machine Learning for CAD (MLCAD)*, pp. 1–6. IEEE, 2023.

Carloni, L. P. From latency-insensitive design to communication-based system-level design. *Proceedings of the IEEE*, 103(11):2133–2151, 2015. doi: 10.1109/JPROC.2015.2480849.

Chang, K., Wang, Y., Ren, H., Wang, M., Liang, S., Han, Y., Li, H., and Li, X. Chipgpt: How far are we from natural language hardware design. *arXiv preprint arXiv:2305.14019*, 2023.

Chang, K., Chen, Z., Zhou, Y., Zhu, W., Wang, K., Xu, H., Li, C., Wang, M., Liang, S., Li, H., Han, Y., and Wang, Y. Natural language is not enough: Benchmarking multi-modal generative AI for Verilog generation. In

Proceedings of the 43rd IEEE/ACM International Conference on Computer-Aided Design, ICCAD '24, pp. 1–9, New York, NY, USA, April 2025. Association for Computing Machinery. ISBN 979-8-4007-1077-3. doi: 10.1145/3676536.3676679.

Clark, C., Lee, K., Chang, M.-W., Kwiatkowski, T., Collins, M., and Toutanova, K. BoolQ: Exploring the Surprising Difficulty of Natural Yes/No Questions. In Burstein, J., Doran, C., and Solorio, T. (eds.), *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 2924–2936, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1300.

Cobbe, K., Kosaraju, V., Bavarian, M., Chen, M., Jun, H., Kaiser, L., Plappert, M., Tworek, J., Hilton, J., Nakano, R., Hesse, C., and Schulman, J. Training Verifiers to Solve Math Word Problems, November 2021.

Cryptography.io. Fernet (symmetric encryption). https://cryptography.io/en/48.0.0/fernet/, 2026. Cryptography 49.0.0.dev1 documentation. Accessed: 2026-05-22.

Dally, W. J., Turakhia, Y., and Han, S. Domain-specific hardware accelerators. *Communications of the ACM*, 63 (7):48–57, 2020.

DeLorenzo, M., Chowdhury, A. B., Gohil, V., Thakur, S., Karri, R., Garg, S., and Rajendran, J. Make Every Move Count: LLM-based High-Quality RTL Code Generation Using MCTS, February 2024a.

DeLorenzo, M., Gohil, V., and Rajendran, J. CreativEval: Evaluating Creativity of LLM-Based Hardware Code Generation. In *2024 IEEE LLM Aided Design Workshop (LAD)*, pp. 1–5, June 2024b. doi: 10.1109/LAD62341.2024.10691798. URL https://ieeexplore.ieee.org/abstract/document/10691798.

Dinella, E., Chandra, S., and Maniatis, P. Crqbench: A benchmark of code reasoning questions. *arXiv preprint arXiv:2408.08453*, 2024. doi: 10.48550/arXiv.2408.08453. URL https://arxiv.org/abs/2408.08453.

Du, M., Tuan, L. A., Ji, B., Liu, Q., and Ng, S.-K. Mercury: A Code Efficiency Benchmark for Code Large Language Models. *Advances in Neural Information Processing Systems*, 37:16601–16622, December 2024. doi: 10.52202/079017-0529.

Duncan, R. G. The role of domain-specific knowledge in generative reasoning about complicated multileveled phenomena. *Cognition and Instruction*, 25(4):271–336, 2007.

Esmaeilzadeh, H., Blem, E., St. Amant, R., Sankaralingam, K., and Burger, D. Dark silicon and the end of multicore scaling. In *Proceedings of the 38th annual international symposium on Computer architecture*, pp. 365–376, 2011.

Fu, W., Yang, K., Dutta, R. G., Guo, X., and Qu, G. Llm4sechw: Leveraging domain-specific large language model for hardware debugging. In *2023 Asian Hardware Oriented Security and Trust Symposium (AsianHOST)*, pp. 1–6. IEEE, 2023.

Gu, A., Roziere, B., Leather, H. J., Solar-Lezama, A., Synnaeve, G., and Wang, S. Cruxeval: A benchmark for code reasoning, understanding and execution. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*, volume 235 of *Proceedings of Machine Learning Research*, pp. 16568–16621. PMLR, July 2024. URL https://proceedings.mlr.press/v235/gu24c.html.

He, X., Liu, Q., Du, M., Yan, L., Fan, Z., Huang, Y., Yuan, Z., and Ma, Z. SWE-Perf: Can Language Models Optimize Code Performance on Real-World Repositories? In *NeurIPS 2025 Fourth Workshop on Deep Learning for Code*, November 2025.

Hendrycks, D., Burns, C., Basart, S., Zou, A., Mazeika, M., Song, D., and Steinhardt, J. Measuring Massive Multitask Language Understanding. In *International Conference on Learning Representations*, October 2020.

Huang, D., Qing, Y., Shang, W., Cui, H., and Zhang, J. EffiBench: Benchmarking the Efficiency of Automatically Generated Code. *Advances in Neural Information Processing Systems*, 37:11506–11544, December 2024. doi: 10.52202/079017-0367.

Huber, P., Aghajanyan, A., Oguz, B., Okhonko, D., Yih, S., Gupta, S., and Chen, X. CCQA: A New Web-Scale Question Answering Dataset for Model Pre-Training. In Carpuat, M., de Marneffe, M.-C., and Meza Ruiz, I. V. (eds.), *Findings of the Association for Computational Linguistics: NAACL 2022*, pp. 2402–2420, Seattle, United States, July 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.findings-naacl.184.

Jimenez, C. E., Yang, J., Wettig, A., Yao, S., Pei, K., Press, O., and Narasimhan, K. R. SWE-bench: Can language models resolve real-world github issues? In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=VTF8yNQM66.

Jin, Q., Dhingra, B., Liu, Z., Cohen, W., and Lu, X. PubMedQA: A Dataset for Biomedical Research Question Answering. In Inui, K., Jiang, J., Ng, V., and Wan, X. (eds.), *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pp. 2567–2577, Hong Kong, China, November 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-1259.

Jung, M., Weis, C., and Wehn, N. Dramsys: A flexible dram subsystem design space exploration framework. *IPSJ Transactions on System and LSI Design Methodology*, 8: 63–74, 2015.

Krieger, S. H. Domain knowledge and the teaching of creative legal problem solving. *Clinical L. Rev.*, 11:149, 2004.

Krishnan, S., Yazdanbakhsh, A., Prakash, S., Jabbour, J., Uchendu, I., Ghosh, S., Boroujerdian, B., Richins, D., Tripathy, D., Faust, A., and Janapa Reddi, V. ArchGym: An Open-Source Gymnasium for Machine Learning Assisted Architecture Design. In *Proceedings of the 50th Annual International Symposium on Computer Architecture*, ISCA '23, pp. 1–16, New York, NY, USA, June 2023. Association for Computing Machinery. ISBN 979-8-4007-0095-8. doi: 10.1145/3579371.3589049.

Lampinen, A., Dasgupta, I., Chan, S., Mathewson, K., Tessler, M., Creswell, A., McClelland, J., Wang, J., and Hill, F. Can language models learn from explanations in context? In *Findings of the Association for Computational Linguistics: EMNLP 2022*, pp. 537–563, 2022.

Lee, S., Kim, S., Park, S., Kim, G., and Seo, M. Prometheus-vision: Vision-language model as a judge for fine-grained evaluation. In *Findings of the association for computational linguistics ACL 2024*, pp. 11286–11315, 2024.

Li, L., Geng, S., Li, Z., He, Y., Yu, H., Hua, Z., Ning, G., Wang, S., Xie, T., and Yang, H. Infibench: Evaluating the question-answering capabilities of code large language models. In *Advances in Neural Information Processing Systems 37 (NeurIPS 2024), Datasets and Benchmarks Track*, 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/e888eb9400fe14bb70e057aa1d719188-Abstract-Datasets_and_Benchmarks_Track.html.

Lin, C.-Y. Rouge: A package for automatic evaluation of summaries. In *Text summarization branches out*, pp. 74–81, 2004.

Liu, M., Ene, T.-D., Kirby, R., Cheng, C., Pinckney, N., Liang, R., Alben, J., Anand, H., Banerjee, S., Bayraktaroglu, I., Bhaskaran, B., Catanzaro, B., Chaudhuri, A., Clay, S., Dally, B., Dang, L., Deshpande, P., Dhodhi, S., Halepete, S., Hill, E., Hu, J., Jain, S., Jindal, A., Khailany, B., Kokai, G., Kunal, K., Li, X., Lind, C., Liu, H., Oberman, S., Omar, S., Pasandi, G., Pratty, S., Raiman, J., Sarkar, A., Shao, Z., Sun, H., Suthar, P. P., Tej, V., Turner, W., Xu, K., and Ren, H. ChipNeMo: Domain-Adapted LLMs for Chip Design, October 2023a.

Liu, M., Pinckney, N., Khailany, B., and Ren, H. Verilogeval: Evaluating large language models for verilog code generation. In *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, pp. 1–8. IEEE, 2023b.

Liu, S., Fang, W., Lu, Y., Zhang, Q., Zhang, H., and Xie, Z. Rtlcoder: Outperforming gpt-3.5 in design rtl generation with our open-source dataset and lightweight solution. In *2024 IEEE LLM Aided Design Workshop (LAD)*, pp. 1–5. IEEE, 2024.

LlamaIndex. Llamaparse: A genai-native document parsing platform. Software, 2025. URL https://cloud.llamaindex.ai/parse. Accessed: 2025-09-18.

Mali, B., Maddala, K., Gupta, V., Reddy, S., Karfa, C., and Karri, R. Chiraag: Chatgpt informed rapid and automated assertion generation. In *2024 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pp. 680–683. IEEE, 2024.

Mañas, O., Krojer, B., and Agrawal, A. Improving automatic vqa evaluation using large language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 4171–4179, 2024.

Meng, X., Srivastava, A., Arunachalam, A., Ray, A., Silva, P. H., Psiakis, R., Makris, Y., and Basu, K. Nspg: Natural language processing-based security property generator for hardware security assurance. In *Proceedings of the 61st ACM/IEEE Design Automation Conference*, pp. 1–6, 2024.

Mhapsekar, K., Ghanbari, A., Aslrousta, B., and Mirbagher-Ajorpaz, S. Cachemind: From miss rates to why-natural-language, trace-grounded reasoning for cache replacement. In *Proceedings of the 31st ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, pp. 307–322, 2026.

Muennighoff, N., Yang, Z., Shi, W., Li, X. L., Fei-Fei, L., Hajishirzi, H., Zettlemoyer, L., Liang, P., Candès, E., and Hashimoto, T. B. s1: Simple test-time scaling. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 20286–20332, 2025.

Nair, M., Sadhukhan, R., and Mukhopadhyay, D. Generating secure hardware using chatgpt resistant to cwes. *Cryptology ePrint Archive*, 2023.

Nguyen, D., Phan, T., Le Hai, N., Doan, T., Nguyen, N., Pham, Q., and Bui, N. CodeMMLU: A Multi-Task Benchmark for Assessing Code Understanding & Reasoning Capabilities of CodeLLMs. *International Conference on Learning Representations*, 2025:2614–2672, May 2025.

OpenAI. Introducing swe-bench verified. Blog post, OpenAI, August 2024. URL https://openai.com/index/introducing-swe-bench-verified/. Updated February 24, 2025.

Opsahl-Ong, K., Ryan, M. J., Purtell, J., Broman, D., Potts, C., Zaharia, M., and Khattab, O. Optimizing Instructions and Demonstrations for Multi-Stage Language Model Programs. In Al-Onaizan, Y., Bansal, M., and Chen, Y.-N. (eds.), *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 9340–9366, Miami, Florida, USA, November 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.emnlp-main.525.

Ouyang, A., Guo, S., Arora, S., Zhang, A. L., Hu, W., Re, C., and Mirhoseini, A. KernelBench: Can LLMs Write Efficient GPU Kernels? In *Forty-Second International Conference on Machine Learning*, June 2025.

Papineni, K., Roukos, S., Ward, T., and Zhu, W.-J. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, pp. 311–318, 2002.

Parashar, A., Raina, P., Shao, Y. S., Chen, Y.-H., Ying, V. A., Mukkara, A., Venkatesan, R., Khailany, B., Keckler, S. W., and Emer, J. Timeloop: A systematic approach to dnn accelerator evaluation. In *2019 IEEE international symposium on performance analysis of systems and software (ISPASS)*, pp. 304–315. IEEE, 2019.

Pearce, H., Tan, B., Ahmad, B., Karri, R., and Dolan-Gavitt, B. Examining zero-shot vulnerability repair with large language models. In *2023 IEEE Symposium on Security and Privacy (SP)*, pp. 2339–2356. IEEE, 2023.

Pei, Z., Zhen, H., Yuan, M., Huang, Y., and Yu, B. BetterV: Controlled Verilog Generation with Discriminative Guidance. In *Forty-First International Conference on Machine Learning*, June 2024.

Phan, L., Gatti, A., Li, N., Khoja, A., Kim, R., Ren, R., Hausenloy, J., Zhang, O., Mazeika, M., Hendrycks, D., Han, Z., Hu, J., Zhang, H., Zhang, C. B. C., Shaaban, M., Ling, J., Shi, S., Choi, M., Agrawal, A., Chopra, A., Nattanmai, A., McKellips, G., Cheraku, A., Suhail, A., Luo, E., Deng, M., Luo, J., Zhang, A., Jindel, K., Paek, J., Halevy, K., Baranov, A., Liu, M., Avadhanam, A., Zhang, D., Cheng, V., Ma, B., Fu, E., Do, L., Lass, J., Yang, H., Sunkari, S., Bharath, V., Ai, V., Leung, J., Agrawal, R., Zhou, A., Chen, K., Kalpathi, T., Xu, Z., Wang, G., Xiao, T., Maung, E., Lee, S., Yang, R., Yue, R., Zhao, B., Yoon, J., Sun, X., Singh, A., Peng, C., Osbey, T., Wang, T., Echeazu, D., Wu, T., Patel, S., Kulkarni, V., Sundarapandiyan, V., Le, A., Nasim, Z., Yalam, S., Kasamsetty, R., Samal, S., Sun, D., Shah, N., Saha, A., Zhang, A., Nguyen, L., Nagumalli, L., Wang, K., Wu, A., Telluri, A., Yue, S., Wang, A., Dodonov, D., Nguyen, T., Lee, J., Anderson, D., Doroshenko, M., Stokes, A. C., Mahmood, M., Pokutnyi, O., Iskra, O., Wang, J. P., Levin, J.-C., Kazakov, M., Feng, F., Feng, S. Y., Zhao, H., Yu, M., Gangal, V., Zou, C., Wang, Z., Popov, S., Gerbicz, R., Galgon, G., Schmitt, J., Yeadon, W., Lee, Y., Sauers, S., Sanchez, A., Giska, F., Roth, M., Riis, S., Utpala, S., Burns, N., Goshu, G. M., Naiya, M. M., Agu, C., Giboney, Z., Cheatom, A., Fournier-Facio, F., Crowson, S.-J., Finke, L., Cheng, Z., Zampese, J., Hoerr, R. G., Nandor, M., Park, H., Gehrunger, T., Cai, J., McCarty, B., Garretson, A. C., Taylor, E., Sileo, D., Ren, Q., Qazi, U., Li, L., Nam, J., Wydallis, J. B., Arkhipov, P., Shi, J. W. L., Bacho, A., Willcocks, C. G., Cao, H., Motwani, S., de Oliveira Santos, E., Veith, J., Vendrow, E., Cojoc, D., Zenitani, K., Robinson, J., Tang, L., Li, Y., Vendrow, J., Fraga, N. W., Kuchkin, V., Maksimov, A. P., Marion, P., Efremov, D., Lynch, J., Liang, K., Mikov, A., Gritsevskiy, A., Guillod, J., Demir, G., Martinez, D., Pageler, B., Zhou, K., Soori, S., Press, O., Tang, H., Rissone, P., Green, S. R., Brüssel, L., Twayana, M., Dieuleveut, A., Imperial, J. M., Prabhu, A., Yang, J., Crispino, N., Rao, A., Zvonkine, D., Loiseau, G., Kalinin, M., Lukas, M., Manolescu, C., Stambaugh, N., Mishra, S., Hogg, T., Bosio, C., Coppola, B. P., Salazar, J., Jin, J., Sayous, R., Ivanov, S., Schwaller, P., Senthilkumar, S., Bran, A. M., Algaba, A., Van den Houte, K., Van Der Sypt, L., Verbeken, B., Noever, D., Kopylov, A., Myklebust, B., Li, B., Schut, L., Zheltonozhskii, E., Yuan, Q., Lim, D., Stanley, R., Yang, T., Maar, J., Wykowski, J., Oller, M., Sahu, A., Ardito, C. G., Hu, Y., Kamdoum, A. G. K., Jin, A., Vilchis, T. G., Zu, Y., Lackner, M., Koppel, J., Sun, G., Antonenko, D. S., Chern, S., Zhao, B., Arsene, P., Cavanagh, J. M., Li, D., Shen, J., Crisostomi, D., Zhang, W., Dehghan, A., Ivanov, S., Perrella, D., Kaparov, N., Zang, A., Sucholutsky, I., Kharlamova, A., Orel, D., Poritski, V., Ben-David, S., Berger, Z., Whitfill, P., Foster, M., Munro, D., Ho, L., Sivarajan, S., Hava, D. B., Kuchkin, A., Holmes, D., Rodriguez-Romero, A., Sommerhage, F., Zhang, A., Moat, R., Schneider, K., Kazibwe, Z., Clarke, D., Kim, D. H., Dias, F. M., Fish, S., Elser, V., Kreiman, T., Vilchis, V. E. G., Klose, I., Anantheswaran,

U., Zweiger, A., Rawal, K., Li, J., Nguyen, J., Daans, N., Heidinger, H., Radionov, M., Rozhoň, V., Ginis, V., Stump, C., Cohen, N., Poświata, R., Tkadlec, J., Goldfarb, A., Wang, C., Padlewski, P., Barzowski, S., Montgomery, K., Stendall, R., Tucker-Foltz, J., Stade, J., Rogers, T. R., Goertzen, T., Grabb, D., Shukla, A., Givré, A., Ambay, J. A., Sen, A., Center for AI Safety, Scale AI, and HLE Contributors Consortium. A benchmark of expert-level academic questions to assess AI capabilities. *Nature*, 649 (8099):1139–1146, January 2026. ISSN 1476-4687. doi: 10.1038/s41586-025-09962-4.

Pinckney, N., Batten, C., Liu, M., Ren, H., and Khailany, B. Revisiting verilogeval: A year of improvements in large-language models for hardware code generation. *ACM Transactions on Design Automation of Electronic Systems*, 2025a.

Pinckney, N., Deng, C., Ho, C.-T., Tsai, Y.-D., Liu, M., Zhou, W., Khailany, B., and Ren, H. Comprehensive verilog design problems: A next-generation benchmark dataset for evaluating large language models and agents on rtl design and verification. *arXiv preprint arXiv:2506.14074*, 2025b.

Press, O., Amos, B., Zhao, H., Wu, Y., Ainsworth, S., Krupke, D., Kidger, P., Sajed, T., Stellato, B., Park, J., Bosch, N., Meril, E., Steppi, A., Zharmagambetov, A., Zhang, F., Pérez-Piñeiro, D., Mercurio, A., Zhan, N., Abramovich, T., Lieret, K., Zhang, H., Huang, S., Bethge, M., and Press, O. AlgoTune: Can Language Models Speed Up General-Purpose Numerical Programs? *Advances in Neural Information Processing Systems*, 38, April 2026.

Qin, J., Xi, Y., Huang, J., Rui, R., Yin, D., Liu, W., Yu, Y., Zhang, W., and Sun, X. APTBench: Benchmarking Agentic Potential of Base LLMs During Pre-Training, October 2025.

Rajpurkar, P., Zhang, J., Lopyrev, K., and Liang, P. SQuAD: 100,000+ Questions for Machine Comprehension of Text. In Su, J., Duh, K., and Carreras, X. (eds.), *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pp. 2383–2392, Austin, Texas, November 2016. Association for Computational Linguistics. doi: 10.18653/v1/D16-1264.

Reddi, V. J. and Yazdanbakhsh, A. Architecture 2.0: Foundations of Artificial Intelligence Agents for Modern Computer System Design . *Computer*, 58(02):116–124, February 2025. ISSN 1558-0814. doi: 10.1109/MC.2024.3521641. URL https://doi.ieeecomputersociety.org/10.1109/MC.2024.3521641.

Rein, D., Hou, B. L., Stickland, A. C., Petty, J., Pang, R. Y., Dirani, J., Michael, J., and Bowman, S. R. GPQA: A Graduate-Level Google-Proof Q&A Benchmark. In *First Conference on Language Modeling*, August 2024.

Rogers, A., Gardner, M., and Augenstein, I. Qa dataset explosion: A taxonomy of nlp resources for question answering and reading comprehension. *ACM Computing Surveys*, 55(10):1–45, 2023.

Sangiovanni-Vincentelli, A. Quo vadis, sld? reasoning about the trends and challenges of system level design. *Proceedings of the IEEE*, 95(3):467–506, 2007. doi: 10.1109/JPROC.2006.890107.

Shypula, A. G., Madaan, A., Zeng, Y., Alon, U., Gardner, J. R., Yang, Y., Hashemi, M., Neubig, G., Ranganathan, P., Bastani, O., and Yazdanbakhsh, A. Learning Performance-Improving Code Edits. In *The Twelfth International Conference on Learning Representations*, October 2023.

Sreedhar, K., Ogbonda, J., Yin, P., Shahidi, N., Nagaraj, K., Deng, Z., Cohen, R., Kalker, T., Kumar, S., Yazdanbakhsh, A., and Subramanian, S. Leveraging LLMs to improve hardware-software co-design workflow productivity and accessibility. In *Machine Learning for Computer Architecture and Systems 2025*, 2025. URL https://openreview.net/forum?id=hPTsqnVzBG.

Sze, V., Chen, Y.-H., Yang, T.-J., and Emer, J. S. Efficient processing of deep neural networks: A tutorial and survey. *Proceedings of the IEEE*, 105(12):2295–2329, 2017.

Thakur, S., Ahmad, B., Pearce, H., Tan, B., Dolan-Gavitt, B., Karri, R., and Garg, S. Verigen: A large language model for verilog code generation. *ACM Transactions on Design Automation of Electronic Systems*, 29(3):1–31, 2024.

Trischler, A., Wang, T., Yuan, X., Harris, J., Sordoni, A., Bachman, P., and Suleman, K. NewsQA: A Machine Comprehension Dataset. In Blunsom, P., Bordes, A., Cho, K., Cohen, S., Dyer, C., Grefenstette, E., Hermann, K. M., Rimell, L., Weston, J., and Yih, S. (eds.), *Proceedings of the 2nd Workshop on Representation Learning for NLP*, pp. 191–200, Vancouver, Canada, August 2017. Association for Computational Linguistics. doi: 10.18653/v1/W17-2623.

Tsai, Y., Liu, M., and Ren, H. RTLFixer: Automatically Fixing RTL Syntax Errors with Large Language Model. In *Proceedings of the 61st ACM/IEEE Design Automation Conference*, DAC '24, pp. 1–6, New York, NY, USA, November 2024. Association for Computing Machinery. ISBN 979-8-4007-0601-1. doi: 10.1145/3649329.3657353.

Tschand, A., Awad, M., Swann, R., Ramakrishnan, K., Ma, J., Lowery, K., Dasika, G., and Reddi, V. J. SwizzlePerf: Hardware-Aware LLMs for GPU Kernel Performance Optimization, 2025.

Vals AI, Inc. Vals.ai benchmarks. https://www.vals.ai/benchmarks, 2025. Accessed: 2025-09-24.

Waghjale, S., Veerendranath, V., Wang, Z., and Fried, D. ECCO: Can We Improve Model-Generated Code Efficiency Without Sacrificing Functional Correctness? In Al-Onaizan, Y., Bansal, M., and Chen, Y.-N. (eds.), *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 15362–15376, Miami, Florida, USA, November 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.emnlp-main.859.

Wan, G.-W., Wong, S., Su, S., Niu, C., Wang, N., Wan, X., Chen, Q., Xing, M., Zhang, J., Ye, J., Wang, Y., Song, R., Ni, T., Xu, Q., Guan, N., Jiang, Z., Wang, X., Chen, Y., and Yang, J. FIXME: Towards End-to-End Benchmarking of LLM-Aided Design Verification. *Proceedings of the AAAI Conference on Artificial Intelligence*, 40(2):1087–1095, March 2026. ISSN 2374-3468. doi: 10.1609/aaai.v40i2.37079.

Wang, X., Li, B., Song, Y., Xu, F. F., Tang, X., Zhuge, M., Pan, J., Song, Y., Li, B., Singh, J., Tran, H., Li, F., Ma, R., Zheng, M., Qian, B., Shao, D., Muennighoff, N., Zhang, Y., Hui, B., Lin, J., Brennan, R., Peng, H., Ji, H., and Neubig, G. OpenHands: An Open Platform for AI Software Developers as Generalist Agents. *International Conference on Learning Representations*, 2025:65882–65919, May 2025.

Wang, Y., Ma, X., Zhang, G., Ni, Y., Chandra, A., Guo, S., Ren, W., Arulraj, A., He, X., Jiang, Z., Li, T., Ku, M., Wang, K., Zhuang, A., Fan, R., Yue, X., and Chen, W. MMLU-Pro: A More Robust and Challenging Multi-Task Language Understanding Benchmark. *Advances in Neural Information Processing Systems*, 37:95266–95290, December 2024. doi: 10.52202/079017-3018.

Wang, Z., Geng, Z., Tu, Z., Wang, J., Qian, Y., Xu, Z., Liu, Z., Xu, S., Tang, Z., Kai, S., Yuan, M., Hao, J., Li, B., and Wu, F. Benchmarking End-To-End Performance of AI-Based Chip Placement Algorithms. *Advances in Neural Information Processing Systems*, 38, April 2026.

Wu, B.-Y., Sharma, U., Kankipati, S. R. D., Yadav, A., George, B. K., Guntupalli, S. R., Rovinski, A., and Chhabria, V. A. EDA Corpus: A Large Language Model Dataset for Enhanced Interaction with OpenROAD. In *2024 IEEE LLM Aided Design Workshop (LAD)*, pp. 1–5, June 2024a. doi: 10.1109/LAD62341.2024.10691774.

Wu, H., He, Z., Zhang, X., Yao, X., Zheng, S., Zheng, H., and Yu, B. Chateda: A large language model powered autonomous agent for eda. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2024b.

Wu, Y. N., Emer, J. S., and Sze, V. Accelergy: An architecture-level energy estimation methodology for accelerator designs. In *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 1–8. IEEE, 2019.

Wulf, W. A. and McKee, S. A. Hitting the memory wall: Implications of the obvious. *ACM SIGARCH computer architecture news*, 23(1):20–24, 1995.

Yang, J., Jimenez, C. E., Wettig, A., Lieret, K., Yao, S., Narasimhan, K. R., and Press, O. SWE-agent: Agent-computer interfaces enable automated software engineering. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://arxiv.org/abs/2405.15793.

Yao, X., Li, H., Chan, T. H., Xiao, W., Yuan, M., Huang, Y., Chen, L., and Yu, B. HDLdebugger: Streamlining HDL debugging with Large Language Models. *ACM Transactions on Design Automation of Electronic Systems*, 30(6):102:1–102:26, October 2025. ISSN 1084-4309. doi: 10.1145/3735638.

Zhang, H., Wang, X., Ao, X., and He, Q. Distillation with explanations from large language models. In *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024)*, pp. 5018–5028, 2024a.

Zhang, Y., Yu, Z., Fu, Y., Wan, C., and Lin, Y. C. Mg-verilog: Multi-grained dataset towards enhanced llm-assisted verilog generation. In *2024 IEEE LLM Aided Design Workshop (LAD)*, pp. 1–5. IEEE, 2024b.

Zheng, L., Chiang, W.-L., Sheng, Y., Zhuang, S., Wu, Z., Zhuang, Y., Lin, Z., Li, Z., Li, D., Xing, E., Zhang, H., Gonzalez, J., and Stoica, I. Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena. *Advances in NeurIPS*, 36:46595–46623, December 2023.

Zhong, H., Xiao, C., Tu, C., Zhang, T., Liu, Z., and Sun, M. JEC-QA: A Legal-Domain Question Answering Dataset. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(05):9701–9708, April 2020. ISSN 2374-3468. doi: 10.1609/aaai.v34i05.6519.

Zhou, C., Liu, P., Xu, P., Iyer, S., Sun, J., Mao, Y., Ma, X., Efrat, A., Yu, P., Yu, L., Zhang, S., Ghosh, G., Lewis, M., Zettlemoyer, L., and Levy, O. LIMA: Less Is More for Alignment. *Advances in Neural Information Processing Systems*, 36:55006–55021, December 2023.

Appendix

Table of Contents

A. Ethics, Reproducibility, and LLM Usage

A.1. Ethics Statement

QUARCH was curated from sources that permit academic use and redistribution. Synthetic items were generated from a domain corpus compiled from public materials, exam-derived items were collected from publicly accessible university course pages or contributed by instructors, and crowdsourced items were submitted through our portal with explicit contributor consent. All expert validators who participated in question review and acceptance are co-authors of this paper. We did not recruit paid crowd workers; when individuals submitted questions via our portal, they consented to inclusion under our dataset license and to public attribution (or opted to remain anonymous). We do not collect personally identifying information beyond optional contact details for acknowledgment. No student data or private repositories were used. Where third-party figures or excerpts are included, we respect the original licenses and provide attribution. We will honor takedown requests for any inadvertently mislicensed content. This project did not involve human-subject experiments or interventions and, to the best of our understanding, does not require IRB oversight.

A.2. Reproducibility Statement

Section 2.2 describes the methodology for constructing QUARCH that can be used to reproduce a dataset of similar quality and characteristics. Appendix D.4, D.5, D.6, D.7, D.8, D.9, and D.10 each expand on the details of the methodology overview provided in Section 2.2. Additionally, exact prompts used for evaluation results are documented in Appendix D.1, D.2, and D.3 for reproducibility.

A.3. LLM Usage

Language models were employed to refine the prose (e.g., grammar, clarity, and style) and to check formatting compliance with venue guidelines. Apart from their explicit roles described in the paper, namely for synthetic QA generation, exam parsing assistance, and evaluation (LLM-as-a-Judge), LLMs were not used to originate substantive scholarly content. All benchmark content admitted to the final release was verified by domain experts, and all prompts used in construction and evaluation are reported in Appendix D.

B. Additional Details & Results

B.1. QUARCH Benchmark and Leaderboard Access

The QUARCH benchmark and leaderboard are available at: `https://quarch.ai/`. Benchmark access is gated to prevent inadvertent inclusion into LLM pretraining corpora and to prevent unauthorized redistribution. Users must agree to conduct evaluations with LLM-as-a-Judge models which are either locally hosted or accessed via APIs which guarantee inputs are never used for training. Our evaluation harness stores ground-truth solutions as Fernet-encrypted (Cryptography.io, 2026) files on disk and only permits in-memory decryption. The QUARCH leaderboard uses Amazon AWS Bedrock APIs for LLM-as-a-Judge evaluation to guarantee solutions are not used for model training.

To further safeguard the benchmark's integrity, we maintain a private, representative hold-out subset of QAs that will remain unreleased. Following the established protocol of seminal benchmarks such as SQuAD (Rajpurkar et al., 2016), researchers must submit their model for evaluation on this hidden test set to qualify for our leaderboard at `https://quarch.ai/`. This serves as a validation step to detect and prevent reporting of inflated results due to data leakage.

B.2. QUARCH Crowdsourcing Website

To provide a centralized location to crowdsource questions and exams, we created the QUARCH website, shown in Figure 7. When a user wants to submit a question, they are presented with a set of instructions to guide accurate, relevant, and formatted questions, shown in Figure 8. Users submit the question, four answer options, the correct answer option, and a rationale for the correct answer, as shown in Figure 9. While the submitted questions are in multiple-choice format, users are instructed to not submit questions where the correct answer is "All of the Above" or "None of the Above." This allows us to concatenate the correct answer with the rationale and format the question as a free-response questions as well.

When a question is written, users can seamlessly test four non-frontier LLMs on correctness, shown in Figure 10. The question and potential answer choices are sent to the respective model using the MCQ Prompt shown in Appendix D.1.

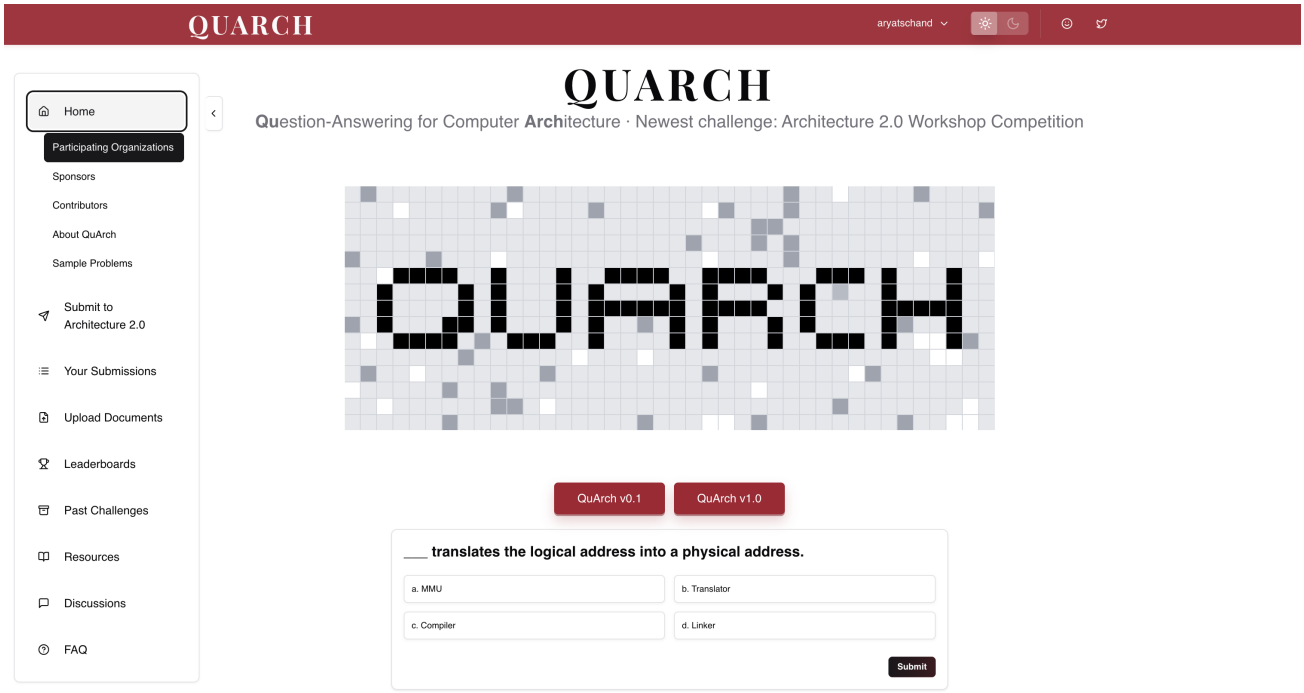


Figure 7. QUARCH Website: Home Page.

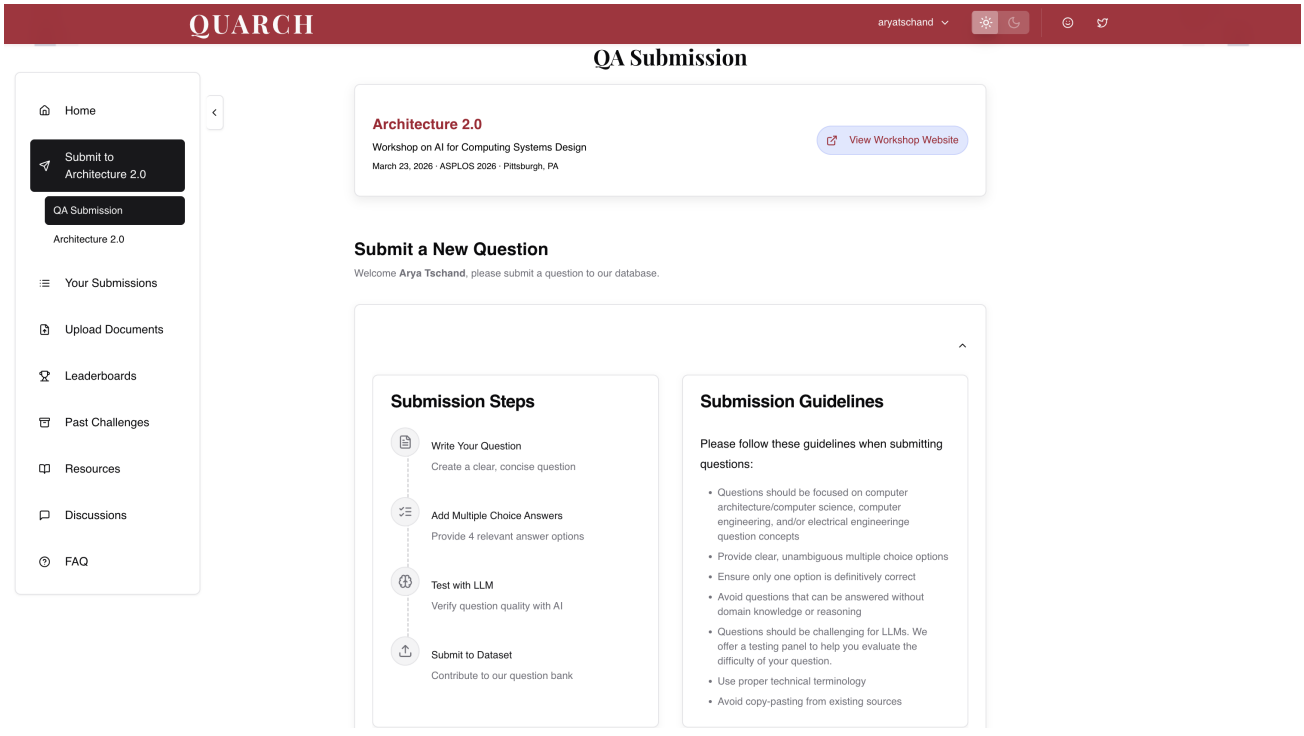


Figure 8. QUARCH Website: QA Submission Instructions.

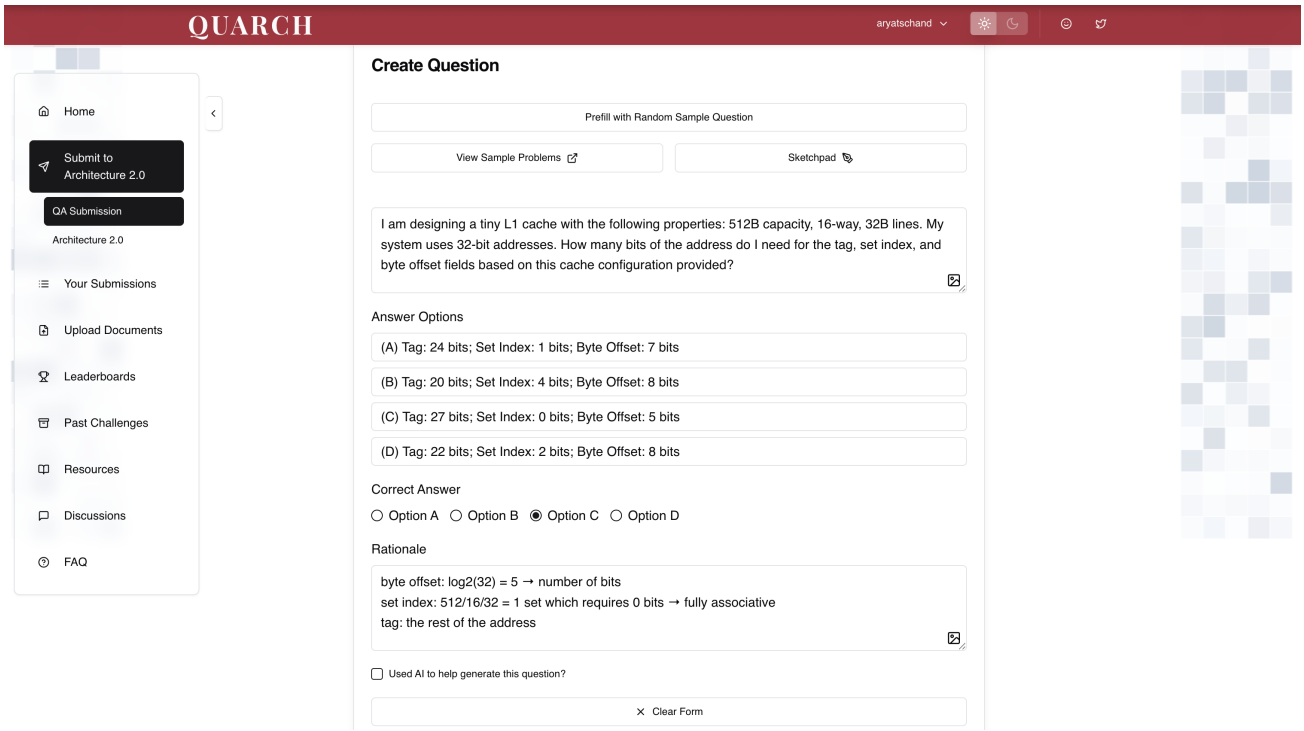


Figure 9. QUARCH Website: QA Submission Portal.

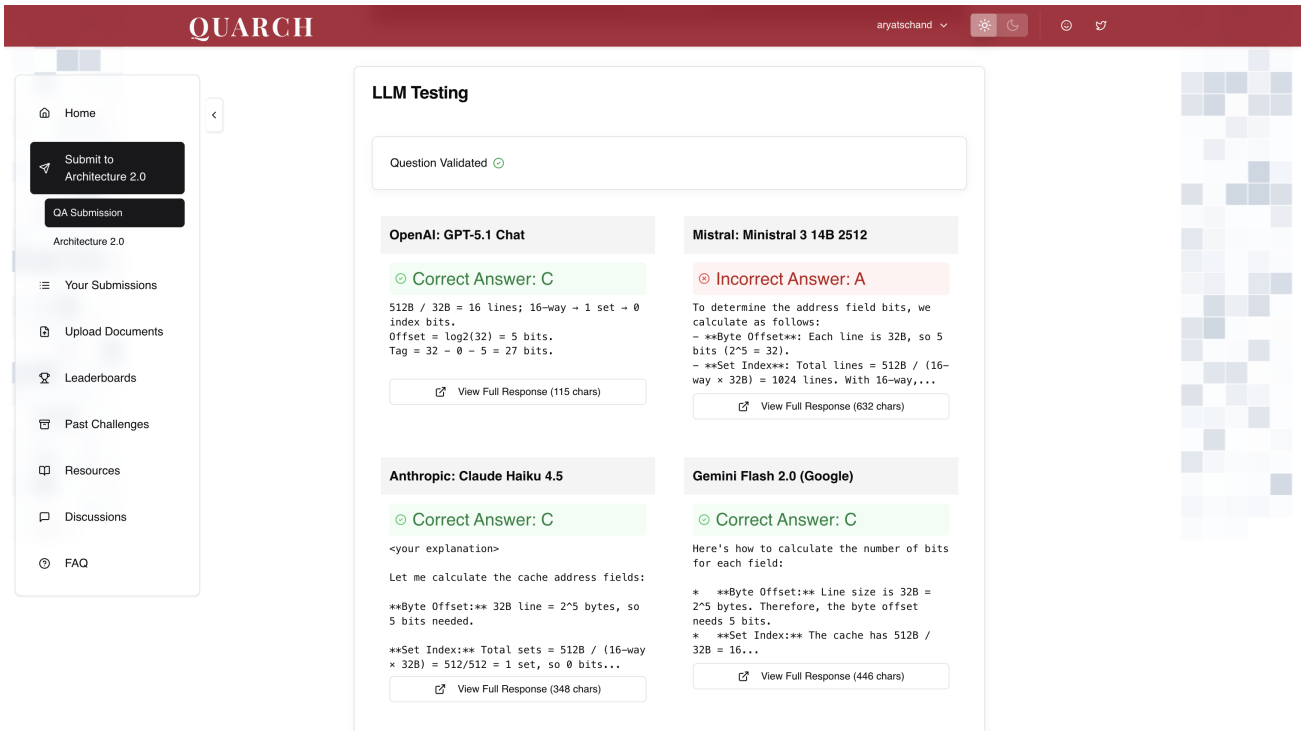


Figure 10. QUARCH Website: LLM Testing on QA Submission.

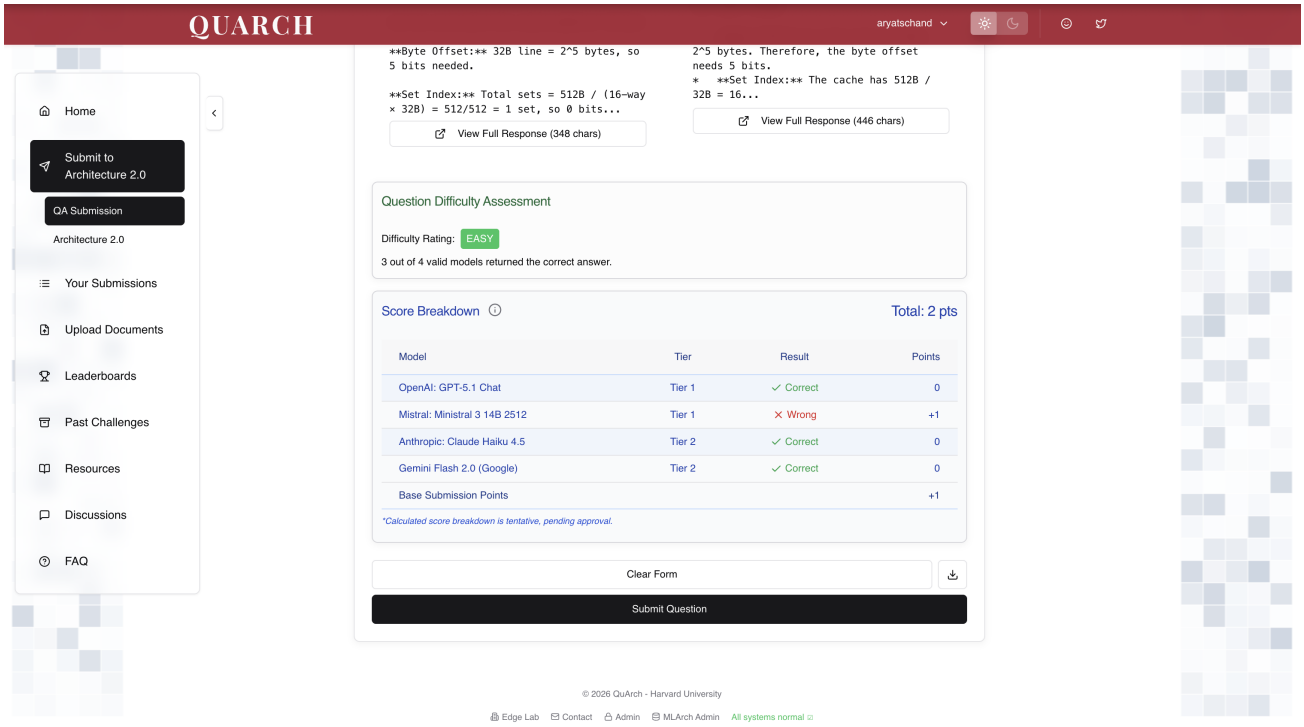


Figure 11. QUARCH Website: QA Submission Scoring.

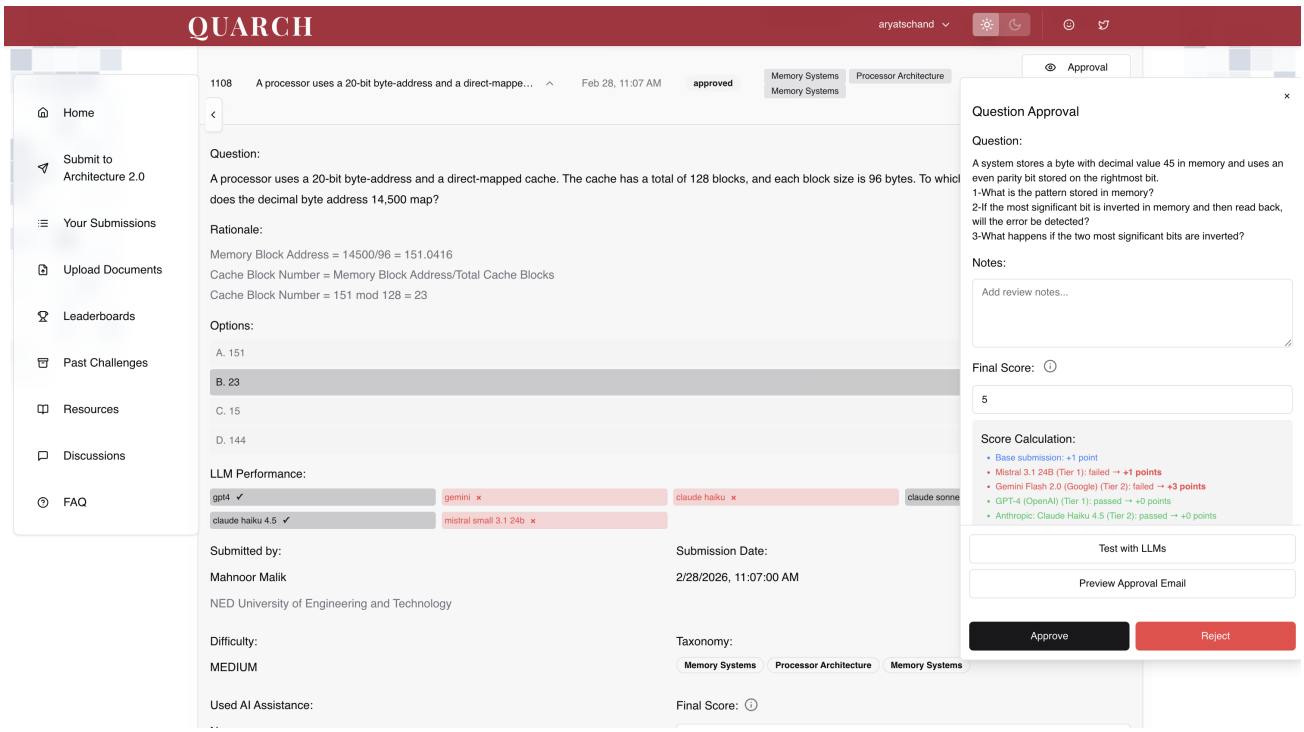


Figure 12. QUARCH Website: Administrator Approval Portal.

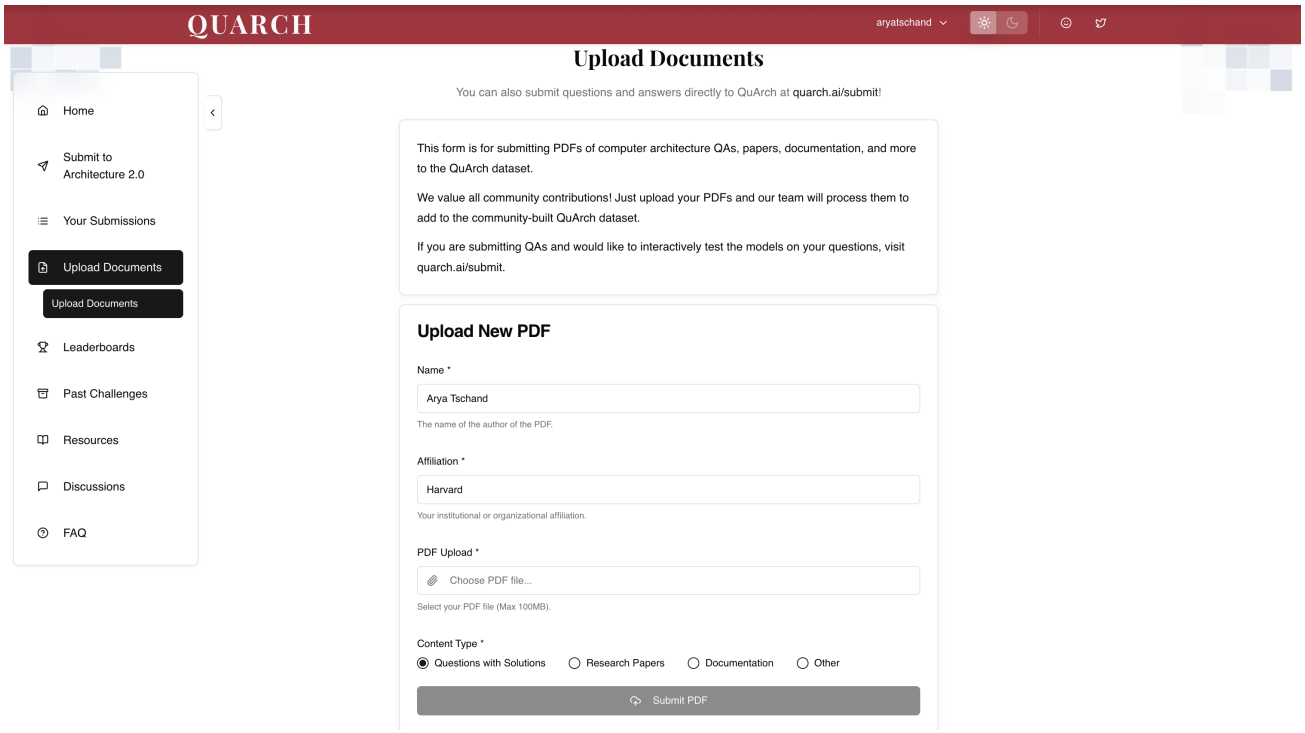


Figure 13. QUARCH Website: Exam Submission Portal.

Users see whether the model gets the question correct and its response. By giving users early signal on whether LLMs fail, they can create difficult and correct questions that challenge a broad range of models. The LLM performance is used to assign a score for the question, where making more frontier models fail scores more points, shown in Figure 11. The website tracks submissions for each user and compiles a leaderboard with the most cumulative points and QA submissions.

When a question is submitted, it is queued for approval in our administrator portal, shown in Figure 12. The approval process involves checking the question for missing assumptions, poor relevance, or insufficient rationale. Administrators also ensure that the QA is objective and correct. Once a submission is approved, the user is notified.

In addition to individual QA submissions, users can submit exams using the website, shown in Figure 13. Questions, context images, and answers are then parsed out from these computer architecture course exams using the offline methodology described in Section 2.2. Finally, the questions collected from the QUARCH crowdsourcing website are taxonomized by skill and funneled into the QUARCH-Reasoning benchmark when appropriate.

B.3. Full Model Results by Skill

Table 6 provides the complete set of results for all evaluated models across the four skill categories in QUARCH. The table illustrates how models perform differently on factual recall compared to higher-order reasoning, design, and implementation. This comprehensive view allows for comparison across both small and large language models, highlighting overall trends and providing a foundation for tracking progress over time. The table captures the broader landscape of model performance and makes clear the varying degrees of capability across skills that are critical for reasoning in computer architecture.

Performance of Small Language Models

Small language models (SLMs) keep pace with LLMs on recall-style questions, but their performance drops on QUARCH-Reasoning QAs, especially when multimodal reasoning is required. This gap suggests that parameter scale (and capacity for long-horizon reasoning and state tracking) matters more for higher-order architectural problem solving (analyze, design, implement) than for factual retrieval. In practice, SLMs are well-suited for low-latency, cost-efficient assistants that handle definitions, quick checks, and targeted lookups, while agentic systems design, trade-off analysis, and figure/table interpretation still benefit from larger models or strong tool scaffolding. A pragmatic path is a cascaded workflow: route

Table 6. Accuracy (%) of all evaluated models across the four skills in QUARCH. Best performing models in each category highlighted **first**, **second**, and **third**.

Model		QUARCH-REASONING			
	Recall	**Analyze**	**Design**	**Implement**	**Overall**
Closed-Source Multimodal Models					
GPT-5.2	88.4	**73.0**	**79.4**	**72.4**	**73.2**
GPT-5	**89.1**	**71.6**	**84.1**	**71.1**	**72.0**
GPT-5 (Non-Reasoning)	86.4	48.9	57.1	39.5	48.4
GPT-4O	83.7	28.1	21.4	22.8	27.4
GEMINI-3-PRO	**90.9**	71.4	**73.8**	65.6	71.0
GEMINI-3-FLASH	**90.1**	**72.8**	66.7	**66.3**	**71.9**
GEMINI-2.5-PRO	87.2	63.1	56.3	63.6	62.9
GEMINI-2.5-FLASH	83.3	57.0	58.7	51.4	56.5
CLAUDE-SONNET-4	85.3	48.8	50.0	41.2	48.1
CLAUDE-3.7-SONNET-THINKING	85.7	52.8	44.4	45.2	51.8
MISTRAL-MEDIUM-3.1	84.2	34.6	38.1	24.5	33.8
Open-Source Multimodal Models					
GEMMA-3-27B-IT	75.3	21.8	**27.0**	15.0	21.4
GEMMA-3-4B-IT	61.6	8.2	4.0	3.1	7.5
QWEN3-VL-32B-NONTHINKING-INSTRUCT	**83.7**	**42.3**	**36.5**	**38.8**	**41.8**
LLAMA-4-MAVERICK	**85.2**	**34.6**	**26.2**	**28.0**	**33.7**
LLAMA-3.2-11B	68.4	9.5	6.3	4.7	9.0
MISTRAL-SMALL-3.2-24B-INSTRUCT	**77.7**	**23.8**	22.2	**18.7**	**23.3**
Text-Only Models					
GPT-OSS-120B	84.5	**63.9**	**66.7**	**58.1**	**63.6**
DEEPSEEK-R1	**86.9**	**56.6**	47.8	44.4	**55.4**
LLAMA-3.3-70B	79.7	27.0	4.3	12.8	25.1
LLAMA-3.2-1B	36.1	2.5	0.0	0.0	2.2
MISTRAL-CODESTRAL-2508	74.6	30.3	14.5	23.9	29.2
MISTRAL-DEVSTRAL-MEDIUM	81.6	28.5	10.1	24.8	27.5
KIMI-K2-0905	84.1	42.8	49.3	37.6	42.7
QWEN3-CODER-480B-A35B-INSTRUCT	82.6	41.6	34.8	25.6	40.2
QWEN3-235B-A22B-THINKING	**85.4**	**62.6**	**60.9**	**45.3**	**61.3**
QWEN3-235B-A22B-NONTHINKING-INSTRUCT	**86.3**	56.1	49.3	**45.3**	55.1
QWEN3-NEXT-80B-A3B-THINKING	84.4	54.5	**50.7**	39.7	53.3
QWEN3-30B-A3B-THINKING	82.3	50.3	42.0	32.1	48.6
QWEN3-CODER-30B-A3B-INSTRUCT	77.6	29.5	15.9	12.0	27.7

recall to SLMs, escalate complex reasoning to LLMs, and bolster SLMs with retrieval and simulators rather than relying on scale alone.

B.4. Topic-Wise Extended Characterization

Figure 14 visualizes the topic-wise performance of the frontier models on QUARCH. As performance across all models on QUARCH-Recall is very high, topic-wise performance variability is not noticeable. However, on the higher-order skills of QUARCH-Reasoning (Analyze, Design, and Implement), models exhibit surprising heterogeneity in per-topic performance.

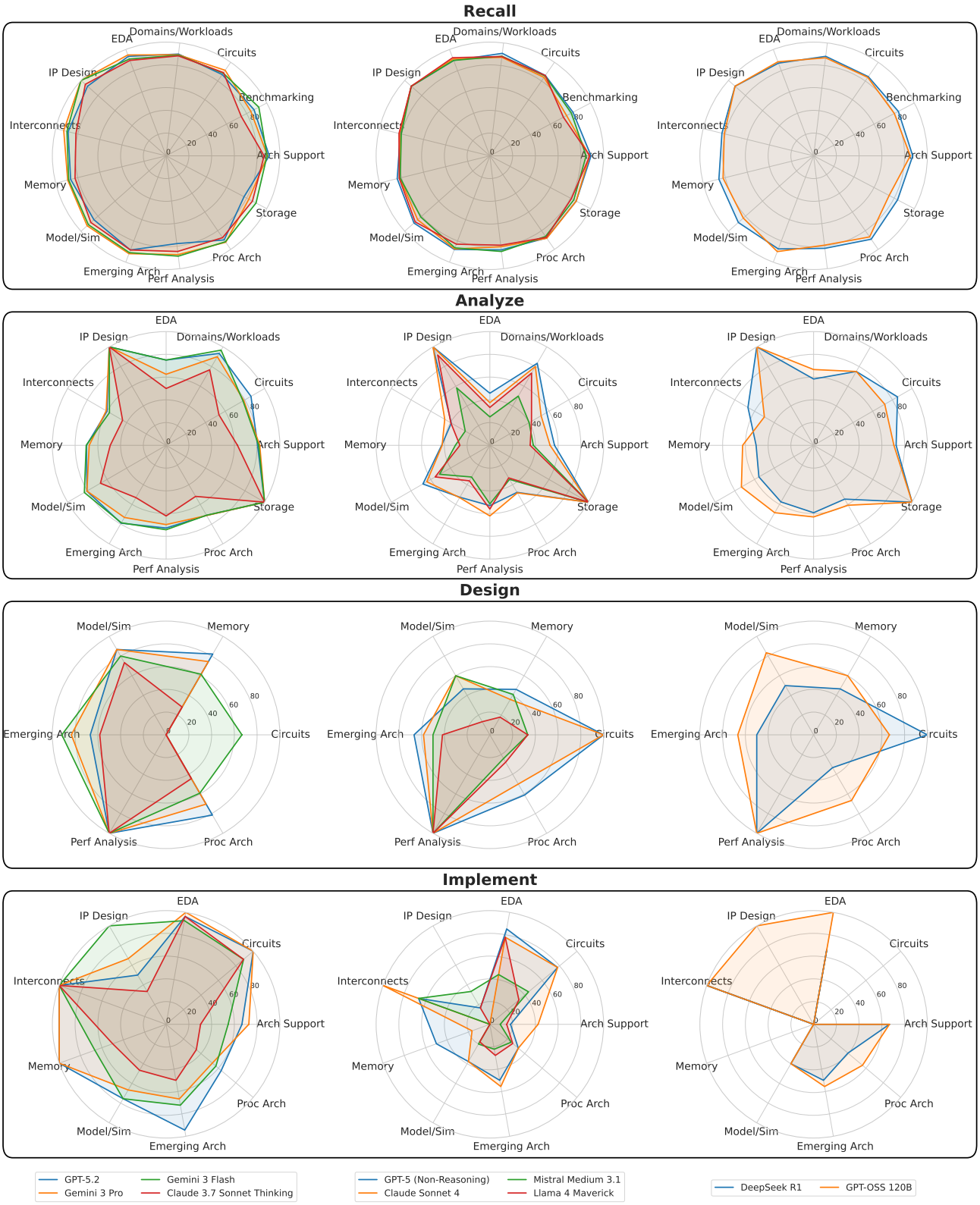


Figure 14. Topic-wise characterization of frontier models, categorized by skill. In order, the groupings of 3 radar plots correspond to "Recall", "Analyze", "Design", and "Implement" questions. Within each grouping, leftmost plot contains the best performing multimodal models, the middle plot contains the worst performing multimodal models, and the rightmost plot contains the two text-only models.

Table 7. Per-generation accuracy (%) by evaluation type and modality. Best performing models in each category highlighted **first**, **second**, and **third**.

Model	Text-only		Multimodal	All
	MCQ	FRQ	FRQ	FRQ
Closed-Source Multimodal Models				
GPT-5.2	88.5	**73.6**	**73.5**	**73.5**
GPT-5	**89.0**	**74.7**	70.5	**72.7**
GPT-5 (Non-Reasoning)	86.4	53.8	44.8	49.4
GPT-4o	84.3	31.3	24.6	28.1
Gemini-3-Pro	**90.9**	71.2	**71.7**	71.4
Gemini-3-Flash	**90.2**	**71.9**	**72.7**	**72.3**
Gemini-2.5-Pro	87.5	63.9	62.1	63.0
Gemini-2.5-Flash	83.2	59.3	55.4	57.4
Claude-Sonnet-4	85.6	52.1	45.4	48.8
Claude-3.7-Sonnet-Thinking	85.9	53.8	51.3	52.6
Mistral-Medium-3.1	84.7	41.2	27.7	34.6
Open-Source Multimodal Models				
Gemma-3-27B-IT	75.7	26.8	**17.7**	22.4
Gemma-3-4B-IT	63.2	9.4	4.1	6.9
Qwen3-VL-32B-NonThinking-Instruct	**84.2**	**44.0**	**38.5**	**41.3**
Llama-4-Maverick	**85.9**	**36.1**	**32.1**	**34.2**
Llama-3.2-11B	70.2	9.5	5.6	8.0
Mistral-Small-3.2-24B-Instruct	**78.6**	**29.1**	**17.7**	**23.5**
Text-Only Models				
GPT-OSS-120B	84.1	**65.3**	-	-
DeepSeek-R1	**87.1**	**56.2**	-	-
Llama-3.3-70B	81.0	23.6	-	-
Llama-3.2-1B	37.1	0.8	-	-
Mistral-Codestral-2508	75.7	28.0	-	-
Mistral-Devstral-Medium	82.5	27.3	-	-
Kimi-K2-0905	84.2	44.2	-	-
Qwen3-Coder-480B-A35B-Instruct	83.3	40.1	-	-
Qwen3-235B-A22B-Thinking	**85.6**	**61.7**	-	-
Qwen3-235B-A22B-NonThinking-Instruct	**86.6**	55.7	-	-
Qwen3-Next-80B-A3B-Thinking	84.5	54.1	-	-
Qwen3-30B-A3B-Thinking	82.7	48.8	-	-
Qwen3-Coder-30B-A3B-Instruct	78.6	26.9	-	-

B.5. Model Performance by Modality

Table 7 performs detailed comparisons between text-only QA performance, image-only performance, and image-text performance across all evaluated models. See Section 4.3 for interpretation and analysis of sensitivity to input modalities. We observe consistent trends across model scales, including significant gaps in higher-order reasoning and multimodal tasks between SLMs and LLMs. By including this wider range of models, we provide a more complete picture of the landscape and enable future work to track progress not only at the frontier but also in more lightweight, cost-efficient models.

Hyperparameters	Config 1	Config 2
SFT Epochs	3	4
Learning Rate	1×10^{-6}	2×10^{-6}
LR Scheduler	Cosine	Cosine
Warmup Ratio	0.30	0.03
Min. LR Ratio	0.00	0.40
Weight Decay	0.00	0.01

Table 8. Hyperparameters used for fine-tuning experiments in Case Study #1.

B.6. Case Study #1: Additional Details and Results

Task Interface and Evaluation Protocol. We first provide further details on the task interface and evaluation protocol used in the case study described in Section 4.4. Each trial consists of an iterative interaction between the language model and simulator, in which the model proposes a memory hierarchy design for a workload. Designs are specified via a structured JSON configuration that encodes the hardware and software decisions listed in Section 4.4. Each proposed configuration is evaluated using the Accelergy tool (Wu et al., 2019) to estimate chip area and energy consumption. After each proposal, the model receives this quantitative feedback on the resulting area and energy, which it may use to refine subsequent designs. Each trial consists of 10 turns to allow the model to iterate on its proposals. A trial is considered successful if the model proposes at least one design within the 10 iterations that consumes at most one millijoule of energy and at most 15,000 μm^2 of chip area; the model is instructed to optimize as much as possible for area while remaining within the energy budget. Importantly, the baseline design provided to the model at the start of each trial does not meet these specifications, requiring models to actively reason about architectural trade-offs in order to meet the design constraints.

Fine-Tuning Details and Additional Results. To construct the fine-tuning dataset, we filtered FRQs from QUARCH related to memory design and matrix multiplication workloads with CLAUDE-3.7-SONNET-THINKING using the filtering prompt listed in D.11. This resulted in 83 questions, 47 of which are text-only. Out of the 47 text-only questions, 45 had at least one correctly-judged response among the benchmarked LLMs in Table 6 and we selected 676 of these responses for our SFT dataset. In particular, 135 of the 676 samples included GPT-5.2 generated explanations of how to arrive at the correct answer, using the three didactic prompts in Section D.11. Our approach to distillation was inspired by Lampinen et al. (2022); Muennighoff et al. (2025); Zhang et al. (2024a). Fine-tuning on all unfiltered FRQs did not provide a similar lift in case study task performance. We hypothesize that for smaller language models, SFT datasets specialized to a single architectural task are simpler to distill from, compared to a diverse mix of closely and distantly related architecture topics, without performing a correspondingly large hyperparameter search.

Table 8 lists the two hyperparameter configurations we used for finetuning GEMMA-3-27B-IT and LLAMA-3.3-70B-INSTRUCT. We used the same 676 SFT samples for both models, and finetuned both models under each set of hyperparameters using Together AI's fine-tuning API.

The first configuration trains each model for 3 epochs, and significantly increases the success rate on the case study task as reported in Table 9. With 4 epochs and a higher learning rate, the second configuration is a more aggressive hyperparameter regime and results in models that are able to discover solutions with much smaller chip area, but at the cost of not significantly improving the success rate (Table 9). The success rate reflects how consistently a model is able to navigate the architectural design space to satisfy strict design constraints, while the best-area metric captures the most area-efficient design discovered across all trials. Together, these results illustrate how fine-tuning on QUARCH questions improves both the reliability and quality of architecture design proposals.

B.7. Case Study #2: Designing a Memory Controller

Case Study #1 (Section 4.4, Appendix B.6) demonstrated knowledge transfer between QUARCH QAs and a downstream cache design task. To futher examine how QUARCH-REASONING performance translates to real architectural decision-making, we conduct an additional case study in this section on Dynamic Random Access Memory (DRAM) controller design, a canonical target of architecture design space exploration tasks (Krishnan et al., 2023). In this task, the model must propose DRAM controller configurations (e.g. scheduling policies, refresh behavior, etc.) that would lead to energy efficient designs. The prompt (Appendix D.12) lists all design parameters and explicitly asks the model to analyze the workload's

Model	Success Rate	Best Area Across Trials (μm^2)
LLAMA-3.3-70B-INSTRUCT		
Base Model	55% (11/20)	7246.78
Fine-tuned Model (Config 1)	**95% (19/20)**	7275.78
Fine-tuned Model (Config 2)	35% (7/20)	**3637.89**
GEMMA-3-27B-IT		
Base Model	30% (6/20)	6728.49
Fine-tuned Model (Config 1)	**70% (14/20)**	6695.84
Fine-tuned Model (Config 2)	50% (10/20)	**3608.89**

Table 9. Results from the Case Study #1 (Section 4.4), grouped by model family. We report the fraction of trials in which each model proposes designs that satisfy both area and energy design constraints (i.e., success rate), along with the best (minimum) chip area discovered across all runs.

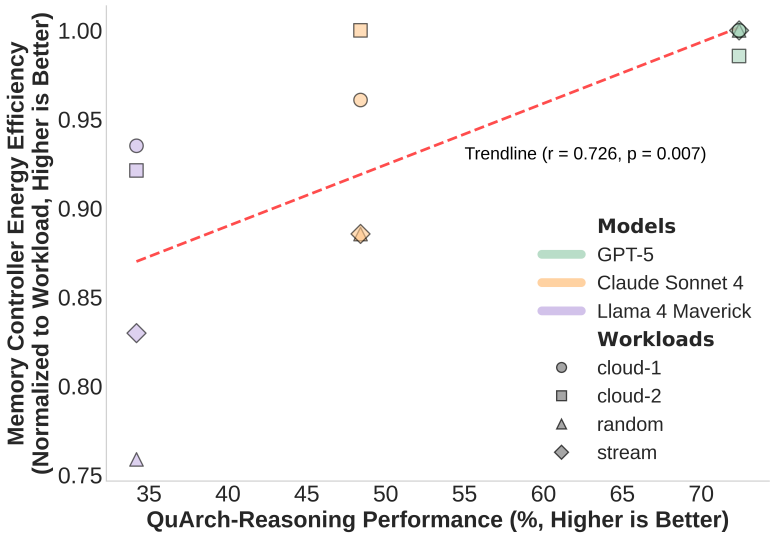


Figure 15. Correlation between QUARCH-REASONING and DRAM controller design quality. Each point is a model-workload pair, with the x-axis showing QUARCH-REASONING accuracy and the y-axis showing normalized memory-controller energy efficiency (power^{-1}).

memory trace and reason about how the controller design choices affect power consumption before proposing a concrete configuration. Critically, (1) effectively answering these questions requires architectural reasoning and influences optimal design choices, and (2) similar QAs about memory trace analysis and memory controller configurations are assessed in QUARCH (Appendix C.7). Designs are evaluated using a DRAM simulator (Jung et al., 2015).

We run 10 independent rounds per model across 4 different workloads under a fixed evaluation budget (40 proposed designs per round). We evaluate three models with varying QUARCH-REASONING performance: GPT-5, CLAUDE-SONNET-4, and LLAMA-4-MAVERICK. Figure 15 reports the mean energy efficiency (power^{-1}, normalized within each workload) of each model's best controller designs across the 10 runs versus their QUARCH-REASONING performance.

We observe a strong and statistically significant positive correlation between QUARCH-REASONING accuracy and normalized energy efficiency ($r = 0.726$, $p = 0.007$), indicating that models which reason better on QUARCH also discover more energy-efficient configurations. This correlation is important, as general QA benchmarks (e.g. MMLU) do not always correlate with downstream task-focused agentic benchmarks (e.g. SWE-Bench) (Qin et al., 2025). This case study finds that improvement on QUARCH directly reflects improvement on computer architecture tasks.

B.8. Partially Correct Judgments

Table 10 reports results when we extend the LLM-as-a-judge rubric to include a "Partially Correct" category. We observe that many models, particularly weaker or smaller ones, produce answers that are not fully correct but demonstrate partial understanding (e.g., identifying the right concept while failing to complete all reasoning steps). Incorporating this intermediate category reveals a richer distribution of model behavior: some models that appear very weak under a strict correct/incorrect

Table 10. Addition of "Partially Correct" Judgments in LLM-as-a-judge rubric. Results are pass@1 (each model was given one attempt at each QA) and using a single LLM-as-a-judge response rather than from consensus.

Model	Correct (%)	Partially Correct (%)	Incorrect (%)
Closed-Source Multimodal Models			
GPT-5.2	74.1	17.6	8.3
GPT-5	70.4	19.2	10.2
GPT-5 (Non-Reasoning)	48.5	31.6	20.0
GPT-4o	28.5	40.2	31.3
Gemini-2.5-Pro	61.9	24.1	13.8
Gemini-2.5-Flash	56.7	26.7	16.5
Claude-Sonnet-4	49.6	31.8	18.6
Claude-3.7-Sonnet-Thinking	52.1	30.7	16.9
Mistral-Medium-3.1	33.5	34.5	31.8
Open-Source Multimodal Models			
Gemma-3-27B-Instruct	24.0	35.2	40.9
Gemma-3-4B-Instruct	6.6	30.9	62.5
Qwen-VL-32B-NonThinking-Instruct	43.4	27.0	29.5
Llama-4-Maverick	33.7	37.9	28.5
Llama-3.2-11B	8.6	27.9	63.5
Mistral-Small-3.2-24B-Instruct	24.5	36.4	39.0
Text-Only Models			
GPT-OSS-120B	66.7	17.5	15.8
DeepSeek-R1	55.9	25.2	18.8
Llama-3.3-70B	24.0	39.5	36.2
Llama-3.2-1B	0.3	13.4	86.3
Mistral-Codestral-2508	27.9	37.7	33.9
Mistral-Devstral-Medium	27.0	37.6	35.2
Kimi-K2-0905	45.4	25.8	28.8
Qwen-3-Coder-480B-Instruct	40.7	34.8	24.5
Qwen-3-235B-A22B-Thinking	57.7	16.6	25.6
Qwen-3-235B-A22B-NonThinking-Instruct	54.4	26.5	18.5
Qwen-3-Next-80B-Thinking	56.1	20.6	23.2
Qwen-3-30B-A3B-Thinking	48.0	21.7	30.1
Qwen-3-Coder-30B-A3B-Instruct	28.9	34.7	36.2

rubric (e.g., sub-30% accuracy) show substantially higher rates of partially correct answers, suggesting they are closer to reaching full correctness than raw accuracy alone fully captures. At the same time, the strongest models still cluster most of their output into "Correct," with only modest use of the partially correct band. This analysis highlights that while partial correctness is less useful in practice for computer architecture tasks that often require precise answers, capturing it can provide a more diagnostic view of model progress and failure modes.

B.9. Comparing Human Domain Experts to LLM-as-a-Judge

In this section, we provide additional details to complement Section 4.5 on validating the fidelity of LLM judges for QuArch QA evaluation, by comparing LLM-as-a-Judge against human expert evaluations of student responses from 10 models considered frontier in September 2025 (GPT-5, GPT-5 (Non-Reasoning), Gemini-2.5-Pro, Gemini-2.5-Flash, Claude-Sonnet-4, Claude-3.7-Sonnet-Thinking, Llama-4-Maverick, Mistral-Medium-3.1, GPT-OSS-120B, and DeepSeek-R1). We instruct LLM-as-a-Judge to reason about the accuracy of each FRQ response with respect to the ground truth answer as though the response is from a student completing an academic exam. See Appendix D.3 for prompts. The judge is instructed to grade each response as Correct, Partially-Correct, or Incorrect. For our reported evaluations (Section 4.1), we recategorize each LLM-as-a-Judge assessment into a binary Correct or Incorrect by rounding down Partially-Correct judge assessments to Incorrect. The Partially-Correct category serves

Table 11. Per-generation accuracy (%) of SoTA models across the four skills of QUARCH using GEMINI-2.5-PRO as the LLM judge. Best performing models in each category highlighted **first**, **second**, and **third**.

Model		QUARCH-REASONING			
	Recall	**Analyze**	**Design**	**Implement**	**Overall**
Multimodal Models					
GPT-5.2	**88.8**	**79.2**	**81.7**	**80.3**	**79.4**
GPT-5 (Non-Reasoning)	86.8	49.5	56.3	42.5	49.1
GEMINI-3-PRO	**91.5**	**79.2**	**82.5**	**81.0**	**79.5**
GEMINI-3-FLASH	**90.7**	**80.0**	**77.0**	**78.9**	**79.8**
CLAUDE-SONNET-4	86.0	51.3	51.6	50.0	51.2
CLAUDE-3.7-SONNET-THINKING	86.1	55.3	53.2	50.7	54.8
MISTRAL-MEDIUM-3.1	84.5	31.1	31.0	27.6	30.8
LLAMA-4-MAVERICK	85.8	33.2	31.0	30.9	32.9
Text-Only Models					
GPT-OSS-120B	84.7	60.9	56.5	56.4	60.4
DEEPSEEK-R1	87.5	58.2	39.1	52.1	57.0

two purposes: it disincentivize the judge from rounding up a nearly-correct answer to correct, and it enables analysis of fine-grained knowledge (Appendix B.8).

We generate multiple samples per question and multiple judgments per sample to control for model stochasticity, and report estimated *pass@k*=1 across 3 samples ($n = 3$) as defined in Pinckney et al. (2025a). For each question in QUARCH, each model under evaluation (student s) generates 3 responses using the model's default generation parameters. For each individual student response, judge model j generates up to 3 assessments until a majority vote consensus is reached. For example, if on a given problem, two student model samples are each judged by majority vote to be CORRECT, and the third student sample is majority vote INCORRECT, *pass@k*=1 on that problem is $\frac{2}{3}$.

Alternative LLM Judges: In this section we justify our selection of CLAUDE-3.7-SONNET-THINKING as the LLM judge used for all benchmark evaluations by comparing Claude against two alternative judge models. We compare Claude against GEMINI-2.5-PRO and QWEN-3-VL-235B-A22B-INSTRUCT as alternate candidates for our LLM judge (using a consensus size of 3 for all three judge models). Across approximately 80,000 responses from 38 student models, we find cross-model agreement rates to closely match (Claude-Gemini: 90.4%, Claude-Qwen: 91.2%, Gemini-Qwen: 89.8%, Table 13). Critically, these rates align with our measured human-to-human agreement rates of 90.92% (Section 4.5), suggesting that LLMs exhibit similar consistency as human experts when judging QAs.

We also compare our three candidate LLM-as-a-Judge models directly against human expert evaluators. Claude had an agreement rate with human judgments of 89.41% (Table 14a), while Gemini and Qwen had slightly lower agreement rates at 88.11% (Table 14b) and 87.03% (Table 14c), respectively.[3] Claude's slightly higher human agreement rate motivates our choice as the primary LLM judge. For completeness, we include an alternative version of Table 3 using Gemini and Qwen as the judge LLM, shown in Appendix Tables 11 and 12.

Majority Adjudication Frequency: We also inspect the frequency of necessitated tie-breaking under our consensus size of 3 (where a tie consists of one CORRECT and one INCORRECT verdict) across QUARCH's entire QA benchmark and all models assessed in the main text and appendix. We observe across 65,659 responses, the first two judgments from CLAUDE-3.7-SONNET-THINKING matched 89.0% of the time (and hence did not require a third judgment to adjudicate).

Fine-Grained Analysis on Self-Judgment Bias: We also seek to investigate whether LLM-as-a-Judge favors responses produced by itself (i.e., if a bias is present when the same model is both the student and the judge). We analyze self-judgment behavior on the human-verified subset of student model responses. In Table 15, we compare CLAUDE-3.7-SONNET-THINKING and GEMINI-2.5-PRO when each model judges its own generations against human expert judgments, grouping

[3]In Table 14, GEMINI-2.5-PRO and QWEN-3-VL-235B-A22B-INSTRUCT use a consensus size of 3 to match our expert human consensus size, but Table 14a for CLAUDE-3.7-SONNET-THINKING only uses single LLM judge generations (i.e., the LLM judgments in Table 14a are not under consensus while the human judgments still are) as this model is no longer available in AWS Bedrock. All other evaluations with CLAUDE-3.7-SONNET-THINKING in this paper use a majority consensus size of 3.

Table 12. Per-generation accuracy (%) of SoTA models across the four skills of QUARCH using QWEN-3-VL-235B-A22B-THINKING as the LLM judge. Best performing models in each category highlighted **first**, **second**, and **third**.

Model		QUARCH-REASONING			
	Recall	**Analyze**	**Design**	**Implement**	**Overall**
Multimodal Models					
GPT-5.2	**88.7**	**77.0**	**80.2**	**82.7**	**77.7**
GPT-5 (Non-Reasoning)	86.8	53.2	59.5	57.1	53.8
GEMINI-3-PRO	**91.1**	**74.5**	**78.6**	**73.8**	**74.6**
GEMINI-3-FLASH	**90.7**	**75.0**	**71.4**	**70.7**	**74.5**
CLAUDE-SONNET-4	85.8	51.6	51.6	53.4	51.7
CLAUDE-3.7-SONNET-THINKING	86.1	53.6	46.0	51.0	53.1
MISTRAL-MEDIUM-3.1	84.7	35.9	45.2	35.4	36.2
LLAMA-4-MAVERICK	85.5	34.1	28.6	30.9	33.6
Text-Only Models					
GPT-OSS-120B	84.8	67.4	75.4	71.8	68.1
DEEPSEEK-R1	87.4	57.8	53.6	53.0	57.3

Table 13. Pairwise confusion matrices comparing LLM-as-a-Judge agreement counts for our three candidate judge models across approximately 80,000 student model responses. Here, agreement denotes that the same judgment ("correct" or "incorrect") is reached by both judge models under majority consensus. Agreement rates between models are broadly similar and mirror human-to-human agreement rates.

(a) CLAUDE-3.7-SONNET-THINKING vs. GEMINI-2.5-PRO

	Gemini	
Claude	Correct	Incorrect
Correct	31,915	3,039
Incorrect	4,549	39,610

(b) CLAUDE-3.7-SONNET-THINKING vs. QWEN-3-VL-235B-A22B-INSTRUCT

	Qwen	
Claude	Correct	Incorrect
Correct	34,541	2,073
Incorrect	5,131	40,233

(c) GEMINI-2.5-PRO vs. QWEN-3-VL-235B-A22B-INSTRUCT

	Qwen	
Gemini	Correct	Incorrect
Correct	33,101	3,339
Incorrect	4,737	37,841

Table 14. Confusion matrices comparing three LLM-as-a-Judge candidate models against human experts on the 925 human-verified student model responses, under majority consensus. CLAUDE-3.7-SONNET-THINKING exhibits a slightly higher agreement rate of 89.41% with human experts compared to the other candidate judge models.

(a) CLAUDE-3.7-SONNET-THINKING vs. Human Experts

	Human Experts	
Claude	Correct	Incorrect
Correct	453	47
Incorrect	51	374

(b) GEMINI-2.5-PRO vs. Human Experts

	Human Experts	
Gemini	Correct	Incorrect
Correct	461	67
Incorrect	43	354

(c) QWEN3-VL-235B-A22B-INSTRUCT vs. Human Experts

	Human Experts	
Qwen	Correct	Incorrect
Correct	468	84
Incorrect	36	337

partially-correct responses with incorrect responses to match our binary FRQ accuracy evaluation. To study potential bias in LLMs, we focus on false positive rate which captures cases where human experts mark a response as incorrect, but LLM-as-a-Judge marks its own response as correct. Claude exhibits a low self-judgment false positive rate of 6.7% (3/45), suggesting that our primary judge does not exhibit substantial self-preference on this subset. In contrast, Gemini exhibits a higher self-judgment false positive rate of 28.6% (8/28), indicating that self-judgment bias can vary across judge models. Together with the cross-judge consistency results in Table 13, this analysis suggests that CLAUDE-3.7-SONNET-THINKING does not exhibit pronounced self-preferential bias.

Grading Difficulty: In order to understand if LLM judge disagreements with human experts increased in frequency for questions that were more challenging to grade, we additionally asked each human expert to assign each QA a score between 1-5 on the difficulty of grading the question (not the difficulty of the question itself). We did not observe any such trend in our data; the domain expertise and familiarity with academic content in the human cohort led to 1 and 2 being the most

Table 15. Confusion matrices comparing self-judgments by LLM-as-a-Judge against human expert judgments on the human-verified subset.

(a) CLAUDE-3.7-SONNET-THINKING Self-Judgments

	Human Experts	
CLAUDE-3.7-SONNET-THINKING	Correct	Incorrect
Correct	48	3
Incorrect	7	42

(b) GEMINI-2.5-PRO Self-Judgments

	Human Experts	
GEMINI-2.5-PRO	Correct	Incorrect
Correct	70	8
Incorrect	2	20

frequently assigned scores for grading difficulty. We believe this preliminary result leads to three potential directions for future work in alignment between the performance of LLM-as-a-Judge and domain experts: (1) judge prompt optimization by both domain experts and automated methods (Opsahl-Ong et al., 2024), such as by informing the judge that students may try to earn extra points on a question they can't answer by including relevant-sounding jargon to mimic understanding as we observe this behavior exhibited by some SLMs, (2) characterizing question difficulty in QUARCH and exploring whether harder questions are also harder to accurately grade by both LLMs and humans, and (3) investigating the tradeoff across LLM judge generation parameters between verdict determinism and verdict accuracy under consensus when comparing against human expert verdicts as ground truth.

C. Example Questions

C.1. QA Skills

C.1.1. EXAMPLE 1: RECALL

Storage Systems

Question: Moving compute closer to the ___ in solid state drives (SSDs) offers higher bandwidth but introduces challenges in managing frequent errors.

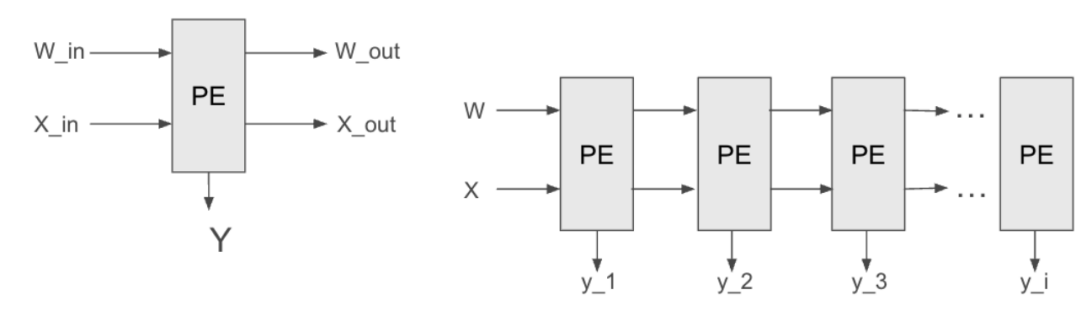
Options:
(a) controller
(b) NAND dies
(c) cache
(d) DRAM

Correct Solution:
(b) NAND dies

Rationale: This question falls under the recall category as it requires retrieval of factual knowledge about SSD architecture, specifically the trade-offs between bandwidth optimization and error management when positioning compute resources relative to different storage components.

C.1.2. EXAMPLE 2: ANALYSIS

Systolic Array



Context: Given arrays $X = [x_1, x_2, x_3, \ldots, x_n]$ and $W = [w_1, w_2, w_3, \ldots, w_k]$, you want to compute $Y = [y_1, y_2, y_3, \ldots, y_{(n+1-k)}]$ using the formula:
$y_i = w_1 x_i + w_2 x_{(i+1)} + w_3 x_{(i+2)} + \cdots + w_k x_{(i+k-1)}$

The figure shows a systolic array of processing elements (PEs) and their input-output behavior.

Question: What should be the relative speeds at which X and W values flow through the array to end up with the correct result for all Y values?

Correct Solution: 2:1

This question is formulated using discussion on Fig 7 in the paper "Why Systolic Architectures?" by HT Kung.

Citation: Kung, "Why systolic architectures?," in Computer, vol. 15, no. 1, pp. 37-46, Jan. 1982, doi: 10.1109/MC.1982.1653825.

Please refer to paper for full discussion.

Intuition below:

The formula for y_i is y_i = w1*x_i + w2*x_(i+1) + w3*x_(i+2) + … + w_k*x_(i+k-1). The number of PEs is equal to the number of Y values. Each PE computes one Y value. A single PE performs the computation Y ← Y + W_in * X_in.

Since w_1 and x_1 are fed as input on the first cycle, the first PE computes y_1. The next PE needs to compute y_2 = w_1*x_2 + w_2*x_3 + ... w_k * x_(k+1). So, after the first cycle, x_2 and w_1 should enter the second PE. This means that x should flow at twice the speed of w to align correctly for the calculations of each y_i. Therefore, the relative speed is 2:1.

Rationale: This is an analysis question because it requires breaking down the systolic array's computational flow and examining how data dependencies between X and W arrays must be synchronized across multiple processing elements.

C.1.3. Example 3: Design

Cache Partitioning and Associativity

Context: Suppose we have a system with 32 cores that share a physical second-level cache. Assume each core is running a single single-threaded application, and all 32 cores are concurrently running applications. Assume that the page size of the architecture is 8KB, the block size of the cache is 128 bytes, and the cache uses LRU replacement. We would like to ensure each application gets a dedicated space in this shared cache without any interference from

other cores. We would like to enforce this using the utility based cache partitioning (UCP) to partition the cache. Assume we would like to design a 4MB cache with a 128-byte block size. Recall that UCP aims to minimize the cache miss rate by allocating more cache ways to applications that obtain the most benefit from more ways, as we discussed in lecture.

Question: Consider the maximum associativity of the cache such that each application is guaranteed a minimum amount of space without interference. Is it desirable to implement UCP on a cache with this maximum associativity? Why, why not? Explain.

Correct Solution: No, it is not desirable to implement UCP with this maximum associativity because the overhead of UCP for 32 applications on this cache will likely outweigh its benefits. UCP will only work with LRU replacement policy. But implementing LRU on top of a 32 k-way cache is impractical. Also the number of counters needed by UCP and the partitioning solution space for UCP are very large for such a cache.

Rationale: This question qualifies as a design question because it requires the analysis and formulation of architectural strategies for cache partitioning in multicore systems. Rather than executing a specific algorithm or implementation, the focus is on evaluating system-level trade-offs, exploring alternative approaches, and proposing optimal solutions under varying associativity constraints.

C.1.4. EXAMPLE 4: IMPLEMENTATION

Linked-List Manipulation via Self-Modifying Code

Context: In this question, you will implement linked-list operations using self-modifying code on an EDSACjr machine. The memory layout is shown in the figure on the right. You have access to the named memory locations as indicated. Linked-list nodes consist of two words: the first is an integer value, the second is an address pointing to the next node. _HEAD contains the address of the first node of the list (or _INVALID if it is empty). The next field of the last node is _INVALID. All valid addresses are positive. You may create new local and global labels as explained in the EDSACjr handout. Table A-1 shows the EDSACjr instruction set.

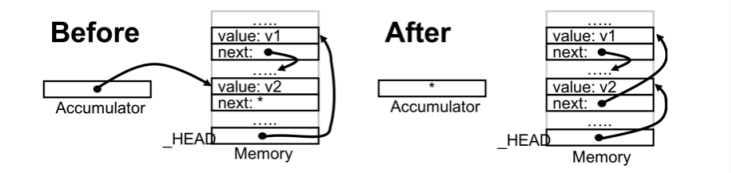
Table A-1 shows the EDSACjr instruction set.

Opcode	Description	Bit Representation
ADD *n*	Accum ← Accum + M[*n*]	00001 *n*
SUB *n*	Accum ← Accum - M[*n*]	10000 *n*
STORE *n*	M[n] ← Accum	00010 *n*
CLEAR	Accum ← 0	00011 00000000000
OR *n*	Accum ← Accum | M[*n*]	00000 *n*
AND *n*	Accum ← Accum & M[*n*]	00100 *n*
SHIFTR *n*	Accum ← Accum shiftr *n*	00101 *n*
SHIFTL *n*	Accum ← Accum shiftl *n*	00110 *n*
BGE *n*	If Accum ≥ 0 then PC ← *n*	00111 *n*
BLT *n*	If Accum < 0 then PC ← *n*	01000 *n*
END	Halt machine	01010 00000000000

Macro	Description
STOREADR *n*	Replace the address field of location *n* with the contents of the accumulator
LOADADR *n*	Load the address field of location *n* into the accumulator

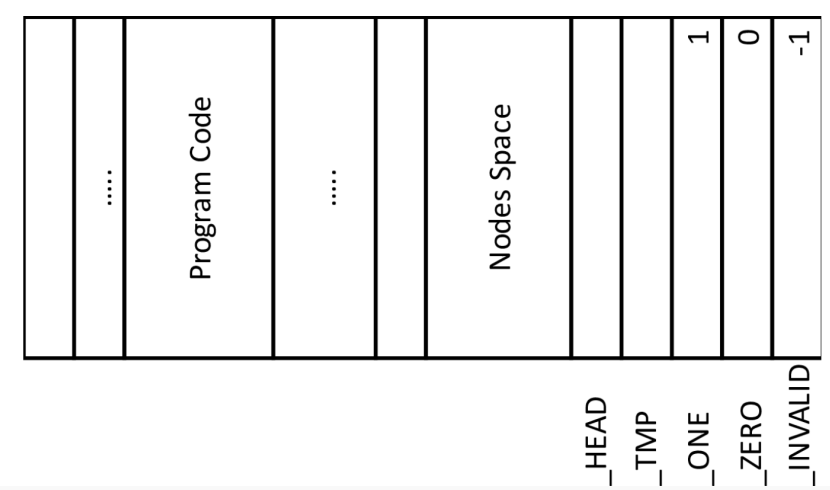
You may also use the following macros if required:

- **STOREADR *n***: Replace the address field of location *n* with the contents of the accumulator
- **LOADADR *n***: Load the address field of location *n* into the accumulator



Write a macro for **LISTPUSH**, which pushes the node pointed to by the accumulator to the head of the list.

LISTPUSH takes one argument, the memory address of the new node, which is available in the accumulator. As shown in the figure below, LISTPUSH stores the current _HEAD pointer in the new node's next field, and updates the _HEAD pointer to point to the new node. Implement the macro using the EDSACjr instruction set and macros provided above. Do not refer to "value" or "next"; they are for illustration only. You need not worry about memory allocation; the new node's address is provided in the accumulator.



Question: Write a macro for LISTPUSH.

```
.macro LISTPUSH
    STORE _TMP   ;; store accumulator (address of the new node)
.end
```

Correct Solution:

```
.macro LISTPUSH
    STORE _TMP    ;; store accumulator (address of the new node)
    ADD _ONE      ;; accum <- address of the new node's next field
    STOREADR _STN ;; address field of location _STN has the address
                  ;; of the new node's next field
    CLEAR
    ADD _HEAD     ;; accum <- M[_HEAD], current head pointer

_STN: STORE 0     ;; 0 will be replaced with the node's next field
                  ;; address. M[_TMP + 1] <- accum
    CLEAR
    ADD _TMP      ;; retrieve address of new node in accumulator
    STORE _HEAD   ;; M[_HEAD] <- accum; Update the head pointer
                  ;; to the new node
.end
```

Rationale: This is an implementation question because it requires constructing actual, working code or macros using the EDSACjr instruction set to perform specific memory operations. Rather than theorizing or analyzing abstract concepts, it requires to translate the algorithm into concrete steps that the hardware can execute.

C.2. Failure Mode #1: Struggles with Architecture-Semantics of Code Execution

C.2.1. EXAMPLE 1

Reuse Intervals in Matrix Multiplication

Context:
For a given compute kernel, we define a tensor's reuse interval (RI) as the number of different elements of that tensor that have been referenced between each re-reference of the same element. For example, consider the following:

```
for m in [0, M)
  for n in [0, N)
    Z[m, n] = A[m] * B[n]
```

Since A's element is used at every iteration of the inner loop, its RI is 1. Each element of B is re-referenced after N references, so its RI is N. Z has "infinite" reuse interval (i.e., no data reuse) since no element is re-referenced throughout the computation:
RI of A = 1
RI of B = N
RI of Z = infinite / no reuse
Consider the following Matrix-Matrix multiply pseudocode, which multiplies two dense matrices A and B to produce Z:

- Multiply two matrices A and B to produce Z
- First matrix A is MxK
- Second matrix B is KxN
- Thus, resulting matrix Z is MxN

```
for m in [0, M)
  for n in [0, N)
    for k in [0, K)
      Z[m, n] += A[m, k] * B[k, n]
```

Question:
What are the reuse intervals for the three matrices? Provide your answers in terms of M, N, and K.

Correct Solution:
RI of A = K
RI of B = N*K
RI of Z = 1

Incorrect Frontier Model Response:
To determine the reuse intervals (RIs) for the matrices A, B, and Z in the given Matrix– Matrix multiply pseudocode, let's analyze the loop structure and how each matrix is accessed.

The pseudocode is as follows:

```
for m in [0, M)
  for n in [0, N)
    for k in [0, K)
      Z[m, n] += A[m, k] * B[k, n]
```

1. Matrix A (MxK).

- The element `A[m, k]` is accessed within the innermost loop (`for k in [0, K)`).
- For a fixed `m`, the row `A[m, k]` is accessed for each `k` from `0` to `K-1`. Thus, for each iteration of the outer loop (`for m in [0, M)`), the same row is not re-referenced until `M` iterations have passed.

- Within the inner loop, each element of a row of `A` is accessed once. So for a given `m`, elements are accessed once as `k` varies from `0` to `K-1`. For the next `m`, the same element is re-referenced after `K` iterations of the inner loop.
- More precisely: the re-reference happens after the inner loop (`for k`) has completed for all `N` iterations of `n`, because `m` is fixed during those `N` iterations. Hence the reuse interval for `A` is `N*K`.
- A finer analysis shows that each element `A[m, k]` is accessed for every `n` in `[0, N)`. Thus, it is reused `N` times before the next `k` is accessed.

Hence, the correct RI for `A` is `N`.
2. Matrix B (KxN).

- `B[k, n]` is accessed within the innermost loop (`for k in [0, K)`).
- For a fixed `n`, `B[k, n]` is accessed for each `k`. Each element of a column is accessed once as `k` varies, and re-referenced after `K` iterations for a different `k`.
- Considering the loop structure: for a given `k` and `n`, `B[k, n]` is accessed for every `m` in `[0, M)`. Thus, it is reused `M` times.

Hence, the RI for `B` is `M`.
3. Matrix Z (MxN).

- `Z[m, n]` is accessed within the innermost loop (`for k in [0, K)`).
- For a fixed `m, n`, `Z[m, n]` is updated `K` times. It is not re-referenced until the computation for that pair is completed, which happens after `K` iterations of the innermost loop.

Hence, the RI for `Z` is `K`.

Therefore, the reuse intervals are:

- RI of A = `N`
- RI of B = `M`
- RI of Z = `K`

These values represent how many times each element of the respective matrices is reused before being re-referenced.

Question Summary and Relevance: This question examines understanding of reuse distance analysis in nested loop kernels, a fundamental concept in computer architecture for predicting cache behavior and memory bandwidth requirements. The reuse interval (RI) counts the number of distinct memory references between successive accesses to the same data element, which affects locality and cache hit rates.

For matrix multiplication, the dominant kernel in scientific computing and deep learning, accurate reuse analysis guides critical optimization decisions such as cache blocking factors, prefetch distances, and loop transformation strategies. The question specifically tests whether models can trace memory access sequences through triple-nested loops and correctly apply the reuse interval definition to determine that A has RI $= K$ (reused after each inner loop completion), B has RI $= N \times K$ (reused only after all $N \times K$ iterations), and Z has RI $= 1$ (immediately reused for accumulation).

Analysis of Model Response: The model's errors reveal a fundamental conceptual confusion between *reuse count* (how many times an element is accessed) and *reuse interval* (distinct accesses between reuses). For example, when analyzing matrix A, the model incorrectly reasons that because "$A[m, k]$ is accessed for every n in $[0, N)$. Thus, it is reused N times...," concluding RI $= N$. This conflates usage frequency with reuse distance; the correct analysis requires tracing the actual reference stream to see that $A[m, k]$ is accessed once, then K different elements are accessed before returning to $A[m, k]$ in the next n iteration, giving RI $= K$. Similarly, for matrix B, the model focuses on B being "reused M times" across different m values, missing that the same $B[k, n]$ element isn't accessed again until after $N \times K$ intervening references.

C.2.2. EXAMPLE 2

Timing Side Channel from Short-Circuit Evaluation

Question:
Consider the code given below:

```
bool detect_duplicate(int* elements, int length){
    bool cond, dup = false;
    for(int i=0; i<length; i++) {
        for(int j=0; j<length; j++) {
            cond = (elements[i] == elements[j]) && (i!=j);
            dup  = CMOV(cond, cond, dup);
        }}
    return !dup;
}
```

Assume that CMOV is implemented as the `cmov` x86 instruction, a secure constant-time instruction. Assume the length of the array is already known publicly and is not sensitive information. Does this code have timing side channels that can reveal insights into the contents of the `elements` array?

Correct Solution:
Yes, there is a timing side channel.

Incorrect Frontier Model Response:

Analysis of Timing Side Channels in the Code:
The provided code aims to detect duplicates in an array without leaking information through timing side channels. Below, I analyze whether the code has timing side channels that could reveal insights into the contents of the `elements` array.

Key Features of the Code.

- The function `detect_duplicate` uses two nested loops that iterate exactly `length` × `length` times, regardless of the array values. This fixed iteration count ensures that the loop structure itself does not introduce timing variations based on the data.
- The critical operation inside the inner loop is a comparison (`elements[i] == elements[j]`) and a logical AND with (`i != j`), followed by a conditional move (CMOV) instruction. The CMOV instruction is specified as a secure constant-time instruction (like the x86 `cmov`), meaning it executes in constant time regardless of the condition outcome.
- The memory access pattern is determined solely by the indices `i` and `j`, which are fixed for a given `length`. Since `length` is public and non-sensitive, the sequence of memory accesses (to `elements[i]` and `elements[j]`) is identical for every invocation with the same `length`. Thus, cache behavior (e.g., hits or misses) is predictable and does not depend on the array values.
- There are no conditional branches that depend on the array values. All instructions (comparisons, logical operations, and CMOV) are executed unconditionally, minimizing the risk of variable execution time due to branch prediction or speculation.

Reasoning.

- Constant-Time Operations: The comparison `elements[i] == elements[j]` is typically implemented as a constant-time integer comparison on modern processors. Similarly, the logical AND and CMOV operations are constant-time under the given assumptions. Therefore, the time taken per inner iteration is invariant to the array values.

- Fixed Workload: The total number of iterations is `length`^{2}, which is constant for a given `length`. Since the work per iteration is constant-time, the overall execution time is determined solely by `length`, not by the array contents.
- Cache Considerations: While cache timing could theoretically vary based on access patterns, here the access pattern is fixed for a given `length`. The values in the array do not influence the addresses accessed, so cache behavior cannot reveal information about the values themselves.
- Return Value: The function returns `!dup`, but this computation occurs after the loops and is trivial (a single negation). It does not affect the timing of the main computation.

Conclusion. Under the assumptions that:

- CMOV is a secure constant-time instruction,
- integer comparisons are constant-time,
- and `length` is public and non-sensitive,

the code does not have timing side channels that can reveal insights into the contents of the `elements` array. The execution time is predictable and depends only on `length`, not on the presence or absence of duplicates or the specific values in the array.

Additional Note. If the processor exhibits timing variations for CMOV or comparisons due to microarchitectural factors (e.g., value-dependent aliasing), these are considered negligible or non-exploitable for side channels in this context, given the problem constraints. However, in absolute terms, constant-time code should be verified on specific hardware, but the code here is designed to be constant-time.

Question Summary and Relevance: This question tests understanding of *timing side-channel vulnerabilities* in security-critical code, a paramount concern in computer security and cryptographic implementations. The problem presents a duplicate detection function that attempts to be constant-time using the `CMOV` (conditional move) instruction, a hardware primitive specifically designed to avoid branch-based timing attacks.

The question is particularly clever because it combines a secure primitive (CMOV) with an insecure pattern (short-circuit evaluation), testing whether one can reason about the interaction between hardware features and high-level programming language semantics. Such vulnerabilities have been exploited in the real-world with attacks against RSA, AES, and other cryptographic implementations.

Analysis of Model Response: The model concluded that the code was secure, citing the constant-time nature of CMOV and the public array length, but it failed to account for the short-circuit evaluation of the logical AND operator in C. This oversight demonstrates that the model failed to effectively reason about the architecture-semantics interplay in code execution. Specifically, the model did not incorporate how code semantics (like conditional evaluation) introduce data-dependent timing variations, leading to an incorrect assessment of side channels.

C.2.3. EXAMPLE 3

SIMD Utilization and Warp Divergence

Context:
We define the SIMD utilization of a program that runs on a GPU as the fraction of SIMD lanes that are kept busy with active threads during the run of a program. As we saw in lecture and practice exercises, the SIMD utilization of a program is computed across the complete run of the program. The following code segment is run on a GPU. A warp in the GPU consists of 64 threads, and there are 64 SIMD lanes in the GPU. Each thread executes a single iteration of the shown loop. Assume that the data values of the arrays A and B are already in vector registers so there are no loads and stores in this program. Both A and B are arrays of integers. (Hint: notice that there are 6 instructions in each thread.)

```
for (i = 0; i < 4096; i++) {
    if (B[i] < 8888) {      // Instruction 1
        A[i] = A[i] * C[i]; // Instruction 2
        A[i] = A[i] + B[i]  // Instruction 3
        C[i] = B[i] + 1;    // Instruction 4
    }

    if (B[i] > 8888) {      // Instruction 5
        A[i] = A[i] * B[i]; // Instruction 6
    }
}
```

Question:
What needs to be true about array B to achieve the minimum possible SIMD utilization?
Show your work. (Please cover all cases in your answer.)

Correct Solution:
1 in every 64 of B's elements are greater than 8888, and 1 in every 64 of B's elements are less than 8888, and the rest of the elements are 8888.

Incorrect Frontier Model Response:
Looking at this problem, I need to analyze how SIMD execution works with conditional branches and determine what values in array B would minimize the number of active threads across all instructions.

Understanding the Code Structure
The code has 6 instructions total:

1. if ($B[i] < 8888$) (condition check)
2. $A[i] = A[i] * C[i]$ (executed if $B[i] < 8888$)
3. $A[i] = A[i] + B[i]$ (executed if $B[i] < 8888$)
4. $C[i] = B[i] + 1$ (executed if $B[i] < 8888$)
5. if ($B[i] > 8888$) (condition check)
6. $A[i] = A[i] * B[i]$ (executed if $B[i] > 8888$)

SIMD Execution with Branches
With 4096 iterations and 64 threads per warp, we have 64 warps total.
In SIMD execution:

- All threads in a warp execute the same instruction simultaneously
- When branches occur, threads not taking a path become inactive
- The warp must execute all paths that any thread takes

Analysis by Cases
For each thread, based on $B[i]$ value:
Case 1: $B[i] < 8888$

- Thread executes: Instructions 1, 2, 3, 4, 5 (instruction 6 skipped)
- Active for 5 out of 6 instructions

Case 2: $B[i] = 8888$

- Thread executes: Instructions 1, 5 only (both conditional blocks skipped)
- Active for 2 out of 6 instructions

Case 3: $B[i] > 8888$

- Thread executes: Instructions 1, 5, 6 (instructions 2, 3, 4 skipped)
- Active for 3 out of 6 instructions

Calculating Active Thread-Instructions per Warp
For a warp with:

- n_1 threads where $B[i] < 8888$
- n_2 threads where $B[i] = 8888$
- n_3 threads where $B[i] > 8888$
- $n_1 + n_2 + n_3 = 64$

Active thread-instructions:

- Instruction 1: 64 threads (all execute condition)
- Instructions 2,3,4: n_1 threads each $= 3n_1$ total
- Instruction 5: 64 threads (all execute condition)
- Instruction 6: n_3 threads

Total per warp $= 64 + 3n_1 + 64 + n_3 = 128 + 3n_1 + n_3$

Minimizing SIMD Utilization
To minimize active thread-instructions, minimize: $3n_1 + n_3$
Since $n_1 + n_2 + n_3 = 64$:

- Set $n_1 = 0$ (no threads with $B[i] < 8888$)
- Set $n_3 = 0$ (no threads with $B[i] > 8888$)
- Set $n_2 = 64$ (all threads with $B[i] = 8888$)

Answer
For minimum SIMD utilization: All elements of array B must equal 8888
Verification

- Total possible thread-instruction slots $= 64 \times 6 \times 64 = 24{,}576$
- With all $B[i] = 8888$: Active thread-instructions $= 64 \times 128 = 8{,}192$
- Minimum SIMD utilization $= \dfrac{8{,}192}{24{,}576} = \dfrac{1}{3} \approx 0.333$

This occurs because when $B[i] = 8888$, threads only execute the two condition checks (instructions 1 and 5) and skip both conditional blocks entirely, minimizing the number of active thread-instructions across the program execution.

Question Summary and Relevance: This problem assess understanding of SIMD utilization and warp divergence in GPUs, a central topic in computer architecture. SIMD (Single Instruction, Multiple Data) execution keeps all lanes busy when threads follow the same path, but utilization drops when conditional branches cause divergence. The question asks what

values in array B minimize utilization across all warps, emphasizing the cost of divergent execution paths. The relevance is twofold: (1) it tests understanding of control flow divergence and its impact on performance and (2) it highlights how data distribution (values in B) directly affects instruction activity. Such reasoning connects hardware execution models to software efficiency, which is a core skill in computer systems design.

Analysis of Model Response: The model response misinterpreted the optimization goal. It assumed that minimizing the *number of active instructions per thread* would directly minimize SIMD utilization, and therefore concluded that setting all $B[i] = 8888$ yields the minimum utilization. However, this reasoning overlooks warp-level execution semantics: if all threads skip the same conditional blocks, the warp does not diverge, and utilization is actually higher (per executed instruction) than in the case where threads diverge.

The core mistake was conflating *per-thread activity minimization* with *warp-level utilization minimization*. The correct answer requires mixing values of B such that different threads in a warp diverge (some with $B[i] < 8888$, some with $B[i] > 8888$, and the rest with $B[i] = 8888$). This forces the warp to execute all branches, but with minimal active participation in each, thereby reducing the overall SIMD utilization.

C.2.4. EXAMPLE 4

Combinational vs. Sequential from Verilog

Question:

```
module one (input clk, input a, input b, output reg [1:0] q);
 always @ (*)
   if (b)
      q <= 2'b01;
   else if (a)
      q <= 2'b10;
endmodule
```

Does the following code result in a sequential circuit or a combinational circuit? Please explain why.

Correct Solution:
This code results in a **sequential** circuit because a latch is required to store the old value of `q` if both conditions are not satisfied.

Incorrect Frontier Model Response:
Answer and Concise Explanation:
This code results in a combinational circuit, but with an important caveat - it creates an unintended latch.

Reasoning:

1. *Sensitivity List Analysis*: The `always @ (*)` statement creates a combinational sensitivity list that triggers whenever any input used within the always block changes. This is the key indicator of combinational logic.
2. *No Clock Edge*: There is no clock edge (like `posedge clk` or `negedge clk`) in the sensitivity list. Although `clk` is declared as an input, it is never used in the always block, so it doesn't affect the circuit behavior.
3. *Data Type Clarification*: The `reg` data type for output `q` does not imply sequential behavior - it's simply a data type that can hold values in procedural blocks.

Critical Issue - Incomplete Case Coverage:

The code has a significant problem:

- When `b = 1`: `q = 2'b01`

- When `b = 0` and `a = 1`: `q = 2'b10`
- When `b = 0` and `a = 0`: No assignment to q

This incomplete case coverage creates an unintended latch because when both inputs are 0, the output `q` retains its previous value, introducing memory behavior.

Conclusion:

While this synthesizes as combinational logic (due to the `always @ (*)` construct), it creates a problematic latch that should be avoided in good design practice. To make it purely combinational, an `else` clause should be added to handle the case when both `a` and `b` are 0.

Question Summary and Relevance: The given Verilog code asks whether the design results in a sequential or combinational circuit. At first glance, the use of `always @(*)` suggests combinational intent since the block triggers on input changes rather than a clock edge. However, because not all input conditions assign a value to the output `q`, the synthesis tool infers a latch to hold the previous value when both `a` and `b` are `0`. This subtlety makes the design sequential, even without an explicit clock.

In the context of computer architecture, understanding such distinctions is critical. Hardware description languages (HDLs) like Verilog are used to describe datapaths, control units, and memory systems. Misclassifying sequential vs. combinational logic can lead to unintended latches, incorrect timing behavior, or hardware inefficiencies. This impacts pipeline stages, critical path delays, and ultimately system performance and reliability, which are important considerations in architecture design.

Analysis of Model Response: The model's response incorrectly concluded that the code produces a purely combinational circuit with an unintended latch. While it correctly identified the incomplete case coverage, it failed to reconcile the key fact that *incomplete assignments within an `always @(*)` block result in latch inference*. This is not "still combinational with a caveat," but a defining feature of sequential logic, since the latch introduces memory behavior.

C.3. Failure Mode #2: Assuming Unconventional Architectural Properties

C.3.1. EXAMPLE 1

Page Table Entry Sizing

Context:
Assume a system which has the following characteristics:

1. Virtual address space is 64 KB and is byte addressable
2. Physical RAM is 16 KB and is byte addressable
3. Page size is 256 Byte
4. One level page table, where each page table entry contains a valid bit, a dirty bit, and the physical page number
5. Integer is 32 bits
6. PTBR is 0x2F5C
7. There is a data TLB that stores two page table entries

Question:
What would be the size of the page table?

Correct Solution:
256 * 8 bits = 256 bytes

Incorrect Frontier Model Response:
Looking at this step-by-step:

Step 1: Calculate the number of virtual pages

Number of virtual pages = Virtual address space ÷ Page size
Number of virtual pages = 64 KB ÷ 256 bytes = 65,536 ÷ 256 = **256 pages**

Step 2: Determine page table entries needed

Since we have a **one-level page table**, we need one page table entry for each virtual page.
Therefore: **256 page table entries** are required.

Step 3: Calculate the size of each page table entry

Each page table entry contains:

- Valid bit: 1 bit
- Dirty bit: 1 bit
- Physical page number: Need to calculate required bits

For the physical page number:

- Physical RAM = 16 KB = 16,384 bytes
- Number of physical pages = 16,384 ÷ 256 = 64 pages
- Bits needed for physical page number = $log_2(64)$ = **6 bits**

Minimum bits per entry = 1 + 1 + 6 = 8 bits

However, the problem states that ”Integer is 32 bits,” which indicates that each page table entry is stored as a 32-bit integer for alignment and efficiency purposes.

Therefore: **Each page table entry = 32 bits = 4 bytes**

Step 4: Calculate total page table size

Total page table size = Number of entries × Size per entry
Total page table size = 256 entries × 4 bytes = **1,024 bytes = 1 KB**

Answer

The size of the page table is **1,024 bytes (1 KB)**.

Question Summary and Relevance: This question asks to calculate the amount of memory required to store a 1-level page table for a system with known virtual and physical memory sizes. This requires calculating the necessary number of bits per page table entry, along with the total number of entries required to represent the entire virtual address space. Additional unnecessary information about the underlying system is provided in the question context, so this question also tests the student's ability to recognize what information is relevant to the question at hand.

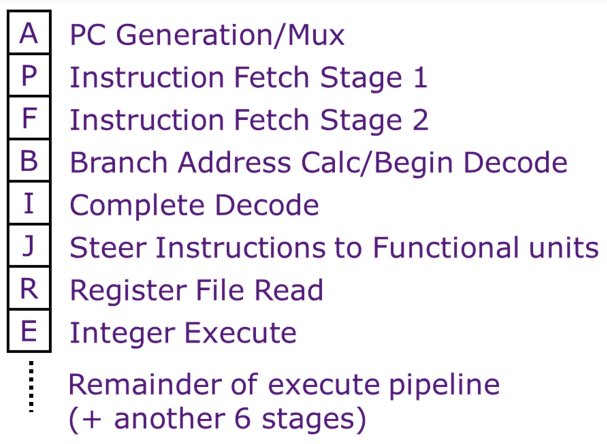
Analysis of Model Response: While the frontier model correctly identifies minimum number of bits required per page table entry (6 physical frame bits + 1 valid bit + 1 dirty bit), it makes an assumption that all page table entries must be aligned to the size of an integer (32 bits). This assumption is unconventional: there is no benefit for aligning page table entries to the

boundary of an unrelated data type. In fact, doing so would leave 24 bits of each page table entry unused and wasted. This assumption causes the model to assume that page table entries must be 32 bits—rather than the required 8 bits—leading the frontier model to incorrectly calculate a required size of 1 KB.

C.3.2. EXAMPLE 2

Branch Misprediction Penalty Cycles

Context:
Ben Bitdiddle is designing a processor with the complex pipeline illustrated below:



The processor has the following characteristics:

- Issues at most one instruction per cycle.
- Branch addresses are known at the end of the B stage (Branch Address Calc/Begin Decode).
- Branch conditions (taken/not taken) are known at the end of the R stage (Register File Read).
- Branches always go through the pipeline without any stalls or queuing delays.

Ben's target program is shown below:

```
for(int i = 0; i <= 1000000; i++)
{
    if(i % 2 == 0) //Branch B1
    { //Not taken
       (Do something A)
    }
    if(i % 4 == 0) //Branch B2
    { //Not taken
       (Do something B)
    }
} //Branch LP



      ANDi R1 0
LOOP: MODi R2 R1 2
      BNE  R2 M4      // B1
      (Do something A)
      ... ...
```

```
M4:   MODi R3 R1 4
      BNE  R3 END     // B2
      (Do something B)
      ... ...
END:  SUBi R4 R1 1000000
      BNE  R4 LOOP    // LP
      ... ...
```

The MODi (modulo-immediate) instruction is defined as follows:

```
MODi Rd Rs imm: Rd <- Rs Mod imm
```

Question:
In steady state, how many cycles per iteration are lost on average if the processor always speculates that every branch is not taken (i.e., next PC is PC+4)?

Correct Solution:
Penalty for miss prediction = 6 cycles
6 * 0.5 + 6 * 0.75 + 6 * 1 = 13.5

Incorrect Frontier Model Response:

Answer: 6.75 cycles per iteration (on average)

Reasoning:

1. Misprediction penalty with "always not taken"
   - Branch target known at end of B.
   - Branch direction known at end of R.
   - If the branch is actually taken but we speculated not taken, redirection waits from B $\rightarrow$ R, i.e., 3 stages (I, J, R).
   - Therefore, penalty per taken branch = 3 cycles.
2. Branch outcomes per iteration
   - B1: i%2==0 $\rightarrow$ BNE is taken when i is odd. Taken 1/2 of iterations.
   - B2: i%4==0 $\rightarrow$ BNE is taken when i%4$\neq$0. Taken 3/4 of iterations.
   - LP (loop back): taken every iteration in steady state.
3. Average cycles lost per iteration
   = 3 $\times$ (P[taken B1] + P[taken B2] + P[taken LP])
   = 3 $\times$ (0.5 + 0.75 + 1)
   = 3 $\times$ 2.25
   = 6.75 cycles/iteration.

Note: "Steady state" ignores the initial and final iterations, where LP is not taken once; including that would change the average negligibly.

Question Summary and Relevance: This question asks the model to calculate the average number of wasted cycles a pipelined processor would incur on a target program when using a simple branch prediction policy. Solving this question requires understanding of the penalty of a branch misprediction on the provided custom processor pipeline, along with reasoning about the average number of mispredictions the target program would incur. Notably, the provided processor pipeline is non-standard, so this question also tests a student's ability to generalize their understanding of the consequences

of branch mispredictions.

Analysis of Model Response: The frontier model correctly understands the average number of branch mispredictions that will occur in the target program (50% of the time for the first `if` statement, 75% of the time for the second `if` statement, and 100% of the time for the `for` loop conditional). However, the model fails to correctly calculate the misprediction penalty, claiming the penalty is 3 cycles instead of 6 cycles. Looking at the reasoning trace, the model incorrectly assumes that during a misprediction, only instructions between stages B and R need to be considered. While this assumption can be valid in simpler pipelines, it doesn't apply here, since instructions in earlier stages (A, P, F) also need to be flushed during a misprediction.

C.4. Failure Mode #3: Modeling and Tracking System State

C.4.1. EXAMPLE 1

Test-and-Set States

Context:
You are writing a queue to be used in a multi-producer/single-consumer application. (Producer threads write messages that are read by one consumer.) We assume here a queue with infinite space. The basic code is shown below.

`TST rs, Imm(rt)` is the test-and-set instruction, which atomically loads the value at Imm(rt) into rs, and if the value is zero, updates the memory location at `Imm(rt)` to 1. This atomic instruction is useful for implementing locks: a value of 1 at the memory location indicates that someone holds the lock, and a value of 0 means the lock is free.

Producer pushes a message onto queue: (memory operations in bold)

```
void push(int** tail_ptr, int* tail_write_lock, int message) {
    while (lock_try(tail_write_lock) == false);
    **tail_ptr = message;
    *tail_ptr++;
    lock_release(tail_write_lock);
}

# R1 - contains address of data to enqueue
# R2 - contains the address of the tail pointer of queue
# R3 - address of tail pointer write lock
P1  SpinLock:TST   R4, 0(R3)  # try to acquire tail write lock
P2  BNEZ           R4, R4, SpinLock
P3  LD             R4, 0(R2)  # get tail pointer
P4  ST             R1, 0(R4)  # write message to tail
P5  ADD            R4, R4, 4  # update tail pointer
P6  ST             R4, 0(R2)
P7  ST             R0, 0(R3)  # release lock

int pop(int** head_ptr, int** tail_ptr) {
    while (*head_ptr == *tail_ptr);
    int message = **head_ptr;
    *head_ptr++;
    return message;
}

# R1 - will receive address contained in message
# R2 - contains the address of the head pointer of queue
```

```
# R3 - contains the address of the tail pointer of the queue
C1 Retry: LD R4, 0(R2)  # get head pointer
C2  LD R5, 0(R3)  # get tail pointer
C3  SUB R5, R4, R5  # is there a message?
C4  BNEZ R5, Pop
C5  JMP Retry
C6 Pop: LD R1, 0(R4)  # read message from queue
C7  ADD R4, R4, 4  # update head pointer
C8  ST R4, 0(R2)
```

Question:
We'll now focus on the tail pointer only. Assuming a MSI invalidate coherence protocol, show the state of the tail pointer in the producer and consumer cache after each operation in the sequence below. Show any data or permissions transfers, e.g. "Memory→C" or "C invalidates P".

How many state transitions occur per message in the steady state?

Operation	Producer tail pointer state	Consumer tail pointer state	Transfers
	I	I	
P1 TST try lock			
P3 LD tail_ptr			
P4 ST message			
P6 ST new_tail			
P7 ST release lock			
C1 LD head_ptr			
C2 LD tail_ptr			
C6 LD message			
C7 ST new_head			
P1 TST try lock			
P3 LD tail_ptr			
P4 ST message			
P6 ST new_tail			
P7 ST release lock			
C1 LD head_ptr			
C2 LD tail_ptr			
C6 LD message			
C7 ST new_head			

How many state transitions occur per message in the steady state?

Correct Solution:

Operation	Producer tail pointer state	Consumer tail pointer state	Transfers
	I	I	
P1 TST try lock			
P3 LD tail_ptr	S		P ← Memory
P4 ST message			
P6 ST new_tail	M		
P7 ST release lock			
C1 LD head_ptr			
C2 LD tail_ptr	S	S	C ← P
C6 LD message			
C7 ST new_head			
P1 TST try lock			
P3 LD tail_ptr			
P4 ST message			
P6 ST new_tail	M	I	P invalidates C
P7 ST release lock			
C1 LD head_ptr			
C2 LD tail_ptr	S	S	C ← P; Memory ← P
C6 LD message			
C7 ST new_head			

How many state transitions occur per message in the steady state? 2 (second half of table)

Incorrect Frontier Model Response:

To analyze the state transitions of the tail pointer in the producer and consumer cache, we need to carefully follow the sequence of operations for both the producer and the consumer, assuming an MSI (Modified, Shared, Invalid) cache coherence protocol.

Steady-State Analysis of Tail Pointer:

Producer Side:

1. P1 TST try lock:
   - State Transition: The producer tries to acquire the lock on the tail pointer.
   - Cache State: If the lock is free (cache line state is I or S), it will transition to M (Modified) when the producer acquires the lock.
2. P3 LD tail_ptr:
   - State Transition: The producer loads the tail pointer into its cache.
   - Cache State: If the tail pointer is not in the cache, it will transition from I (Invalid) to S (Shared) or E (Exclusive) if it is not shared.
3. P4 ST message:
   - State Transition: The producer writes the message to the tail location.
   - Cache State: This write will transition the cache line to M (Modified).
4. P6 ST new_tail:
   - State Transition: The producer updates the tail pointer.
   - Cache State: The tail pointer cache line will transition to M (Modified) in the producer's cache.

5. P7 ST release lock:
    - State Transition: The producer releases the lock.
    - Cache State: The lock cache line will transition back to I (Invalid) or S (Shared) depending on other cores' states.

Consumer Side:

1. C2 LD tail_ptr:
    - State Transition: The consumer loads the tail pointer into its cache.
    - Cache State: If the tail pointer is not in the cache or is invalid, it will transition from I to S (Shared) or E (Exclusive).

—

MSI State Transitions per Message in Steady State:

1. Producer Acquires Lock (P1 TST try lock):
    - Transition from I or S to M (1 transition).
2. Producer Loads Tail Pointer (P3 LD tail_ptr):
    - Transition from I to S or E (1 transition).
3. Producer Writes Message (P4 ST message):
    - Transition from S or E to M (1 transition).
4. Producer Updates Tail Pointer (P6 ST new_tail):
    - Transition from S or E to M (1 transition).
5. Producer Releases Lock (P7 ST release lock):
    - Transition from M to I (1 transition).
6. Consumer Loads Tail Pointer (C2 LD tail_ptr):
    - Transition from I to S (1 transition), potentially causing an invalidation in the producer's cache (1 additional transition).

—

Data/Permission Transfers: - Memory to Cache (Memory $\rightarrow$ C): When the tail pointer is initially loaded into the cache. - Cache Invalidation (C invalidates P): When the consumer loads the tail pointer, it may invalidate the producer's copy if it was in M state.

—

Summary of State Transitions per Message:

1. Producer:
    - Acquire lock: I/S $\rightarrow$ M (1 transition)
    - Load tail pointer: I $\rightarrow$ S/E (1 transition)
    - Write message: S/E $\rightarrow$ M (1 transition)
    - Update tail pointer: S/E $\rightarrow$ M (1 transition)
    - Release lock: M $\rightarrow$ I (1 transition)
2. Consumer:
    - Load tail pointer: I $\rightarrow$ S (1 transition)
    - Invalidate producer's tail pointer: M $\rightarrow$ I (1 transition)

Total State Transitions per Message:

- 5 transitions on the producer side.
- 2 transitions on the consumer side.

Total: 7 state transitions per message in the steady state.

Operation	Producer Cache State Transition	Consumer Cache State Transition	Transfers
P1 TST try lock	I/S → M	-	-
P3 LD tail_ptr	I → S/E	-	Memory → P
P4 ST message	S/E → M	-	-
P6 ST new_tail	S/E → M	-	-
P7 ST release lock	M → I	-	-
C2 LD tail_ptr	-	I → S	Memory → C, C invalidates P

—

Conclusion: In the steady state, there are 7 state transitions per message considering both the producer and consumer operations. Each message involves multiple cache line state changes and potential invalidations to maintain coherence.

Question Summary and Relevance: This question assesses whether models can concretely reason about cache coherence behavior in a multicore setting. By stepping through the producer–consumer interaction under an MSI protocol, it forces careful tracking of how load, store, and lock operations map to specific cache-line state transitions and invalidations. The question relies on architectural intuition – seeing how simple pointer updates or lock releases induce multiple coherence transitions highlights the hidden costs of synchronization and communication, which often create a scalability bottleneck in shared-memory systems.

Analysis of Model Response: The model incorrectly asserted that the producer's test-and-set (TST) on the lock variable alters the MSI state of the tail pointer. This reflects a misunderstanding of coherence granularity: the MSI protocol tracks cache lines at the level of physical addresses, not program-level abstractions. Since the lock and the tail pointer occupy distinct memory locations, a TST on the lock can only trigger transitions on the lock's cache line. Attributing such transitions to the tail pointer indicates a conceptual error in tracking the architectural system state, where ordering effects of synchronization are mistakenly interpreted as coherence events on unrelated data.

C.4.2. EXAMPLE 2

Identifying Data Dependencies from Instruction Trace

Context:

```
addi x2, x0, 17
addi x8, x0, 14
add x1, x1, x2
and x3, x4, x1
addi x1,x2, 5
add x3, x3, x3
sw x3, 4(x1)
add x7, x5, x4
sw x7, 0(x5)
add x11, x2, x3
add x9, x7, x8
add x9, x9, x9
ld x11, 12(x8)
ld x9, 0(x5)
```

Question:
Consider the above RISC-V instruction sequence. The total number of instruction pairs having Read after Write (RAW) dependencies on registers and memory is ___?

Correct Solution:
There are 13 RAW dependencies on registers and 2 RAW dependencies on memory (for memory addresses 0(x5) and 26)

Incorrect Frontier Model Response:

I need to analyze this RISC-V instruction sequence to identify all Read-after-Write (RAW) dependencies on both registers and memory.
Let me work through this step-by-step, numbering each instruction and tracking dependencies:

Instructions:

```
addi x2, x0, 17    # x2 = 0 + 17 = 17
addi x8, x0, 14    # x8 = 0 + 14 = 14
add x1, x1, x2     # x1 = x1 + x2
and x3, x4, x1     # x3 = x4 & x1
addi x1, x2, 5     # x1 = x2 + 5
add x3, x3, x3     # x3 = x3 + x3
sw x3, 4(x1)       # store x3 to memory[x1 + 4]
add x7, x5, x4     # x7 = x5 + x4
sw x7, 0(x5)       # store x7 to memory[x5 + 0]
add x11, x2, x3    # x11 = x2 + x3
add x9, x7, x8     # x9 = x7 + x8
add x9, x9, x9     # x9 = x9 + x9
ld x11, 12(x8)     # load x11 from memory[x8 + 12]
ld x9, 0(x5)       # load x9 from memory[x5 + 0]
```

RAW Dependencies on Registers:
Instruction 3 reads x2 (written by instruction 1) → RAW: (1,3)
Instruction 4 reads x1 (written by instruction 3) → RAW: (3,4)
Instruction 5 reads x2 (written by instruction 1) → RAW: (1,5)
Instruction 6 reads x3 (written by instruction 4) → RAW: (4,6)
Instruction 7 reads x3 (written by instruction 6) → RAW: (6,7)
Instruction 7 reads x1 (written by instruction 5) → RAW: (5,7)
Instruction 9 reads x7 (written by instruction 8) → RAW: (8,9)
Instruction 10 reads x2 (written by instruction 1) → RAW: (1,10)
Instruction 10 reads x3 (written by instruction 6) → RAW: (6,10)
Instruction 11 reads x7 (written by instruction 8) → RAW: (8,11)
Instruction 11 reads x8 (written by instruction 2) → RAW: (2,11)
Instruction 12 reads x9 (written by instruction 11) → RAW: (11,12)
Instruction 13 reads x8 (written by instruction 2) → RAW: (2,13)

RAW Dependencies on Memory:
Instruction 14 reads from memory[x5 + 0] (written by instruction 9) → RAW: (9,14)

Summary:
Register RAW dependencies: 13 pairs
Memory RAW dependencies: 1 pair

Total RAW dependencies: 14
The answer is 14.

Question Summary and Relevance: This question evaluates a student's ability to identify and count Read-After-Write (RAW) dependencies in a given RISC-V instruction sequence, both at the register level and the memory level. All register dependencies and some memory dependencies can be found by directly statically analyzing the instructions in isolation. However, one memory dependency (address 26) requires properly tracking the values of registers `x8` and `x1`.

Analysis of Model Response: The model successfully tracked register states but missed the more hidden memory RAW dependency on address 26. This limitation led to errors in identifying Read-after-Write (RAW) dependencies and ultimately produced an incorrect answer, illustrating the model's weakness in handling memory-based interactions and implicit architectural information.

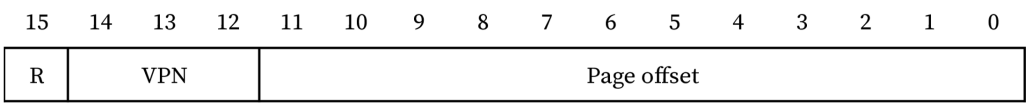
C.5. Failure Mode #4: Sensitivity to QA Modality

C.5.1. EXAMPLE 1

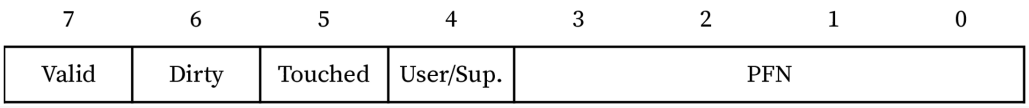
Interpreting Bit-vectors for Page Table Entries

Context:
A processor supports virtual memory with two regions (user and supervisor) using a VAX-like address translation approach. A breakdown of a virtual address and a page-table entry are shown below. Use these to answer each of the following questions.

Virtual address:



Page table entry (PTE):



Question:
What is the page size?

Solution:
4KiB (12 offset bits)

Incorrect Frontier Model Response:

Answer: Page Size Calculation

To determine the page size in bytes, I need to examine the structure of the virtual address and identify how many bits are allocated to the page offset.

Analysis of Virtual Address Structure

From the provided virtual address format:

- Bit 15: R (Region bit)
- Bits 14-9: VPN (Virtual Page Number)

- Bits 8-0: Page offset

Calculation

The page offset field determines the size of each page in the memory system. With a page offset of 9 bits (bits 8 through 0), the page size can be calculated as:

Page size = $2^{\text{number of page offset bits}}$ bytes
Page size = 2^9 bytes
Page size = 512 bytes

Therefore, the page size in this virtual memory system is 512 bytes.

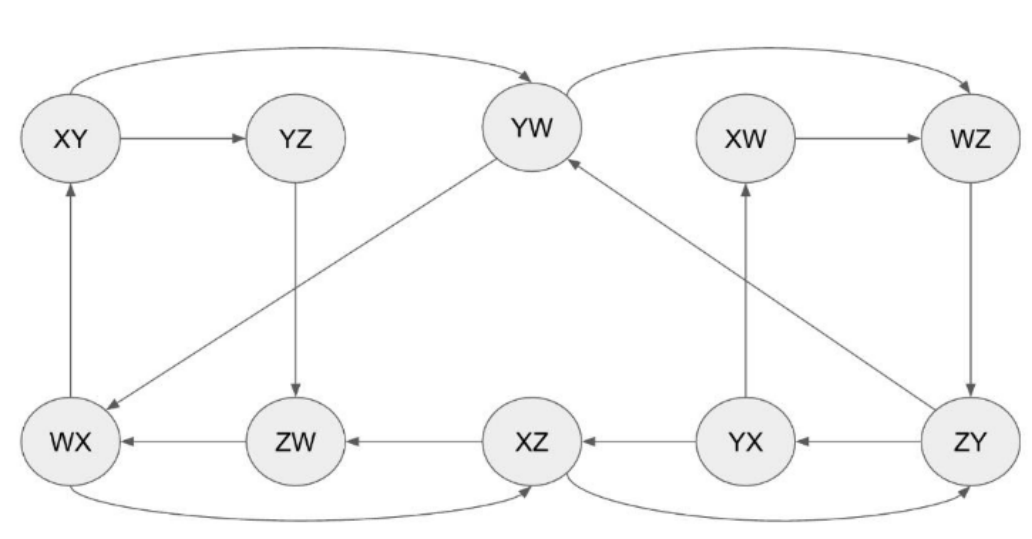
Question Summary and Relevance: This question asks the student to interpret a bit-vector of a virtual address to calculate the appropriate page size. This requires understanding that page sizes are dependent on the size of the offset region of a virtual address. For this question, the bit-vector for a virtual address is provided as an image.

Analysis of Model Response: Here, the model failed to properly interpret the provided image containing the bit-vector of a virtual address. While the image clearly shows bits 11:0 containing the page offset, the frontier model incorrectly claims the offset is in bits 8:0. Importantly, the model does execute the correct calculation for the size of a page based on the offset bit length, so this model failure was purely in the interpretation of the provided image.

C.5.2. EXAMPLE 2

Deadlock-Free Channel Dependency Graphs

Question:
The image shows a channel dependency graph for a network. What is the smallest set of edges that, when removed, eliminates deadlocks in the network?



A. $ZW \to WX, ZY \to YX$

B. $ZW \to WX, ZY \to YX, YW \to WX, ZY \to YW$

C. $YW \to WX, ZY \to YW, XY \to YZ, XW \to WZ$

D. $ZW \to WX$

Correct Solution:
Eliminating edges mentioned in Option B removes all cycles from the graph and hence eliminates deadlocks. Since there are 8 cycles in the given graph, we need to eliminate at least 4 edges.

Incorrect Frontier Model Response:
Looking at this channel dependency graph, I need to identify the smallest set of edges that, when removed, eliminates all deadlocks.

First, let me identify the cycles in the graph, as deadlocks occur when there are circular dependencies:

- XY → YZ → ZW → WX → XY (4-node cycle)
- YW → WX → XZ → ZY → YW (4-node cycle)
- XW → WZ → ZY → YX → XW (4-node cycle)

To eliminate deadlocks, I need to break all cycles. The key is finding the minimum number of edges that participate in multiple cycles, as removing such edges breaks multiple cycles simultaneously.

Let me analyze which edges appear in multiple cycles:

- ZW→WX: Appears in cycles 1 and 2
- ZY→YX: Appears in cycles 2 and 3
- YW→WX: Appears in cycle 2
- XZ→ZY: Appears in cycles 2 and 3

Looking at the options:

- ZW→WX, ZY→YX: This removes edges that break all three cycles.
  Removes ZW→WX (breaks cycles 1 and 2).
  Removes ZY→YX (breaks cycles 2 and 3).
  All cycles are broken with just 2 edges.
- ZW→WX, ZY→YX, YW→WX, ZY→YW: This is overkill (4 edges).
- YW→WX, ZY→YW, XY→YZ, XW→WZ: This is also overkill (4 edges).
- ZW→WX: This only breaks cycles 1 and 2, leaving cycle 3 intact.

The smallest set that eliminates all deadlocks is ZW→WX, ZY→YX (option 1), as it breaks all cycles with just 2 edges.

The answer is the first option: ZW→WX, ZY→YX.

Question Summary and Relevance: The question tests students' knowledge of deadlock freedom conditions and ability to interpret directed dependency graphs. Since the graph has quite a few nodes (10 nodes), a visual representation of the dependency graph is the most natural way to represent this setting.

Analysis of Model Response: The model failed to properly track the cycles in the graph and performed an oversimplified traversal, ignoring edge cases/cycles where deadlock could have occurred. This likely is due to a failure to track the cycles via the visual representation of the dependency graph.

C.6. Example Motivating LLM-as-a-Judge for Automated Evaluations

Advantages of LdR/StC pair of instructions

Context:
One of the common instruction sequences used for synchronizing several processors are the LOAD RESERVE/STORE CONDITIONAL pair (from now on referred to as LdR/StC pair). The LdR instruction reads a value from the specified address and sets a local reservation for the address. The StC attempts to write to the specified address provided the local reservation for the address is still held. If the reservation has been cleared the StC fails and informs the CPU.

Question:
Give two reasons why the LdR/StC pair of instructions is preferable over atomic read-test- modify instructions such as the TEST & SET instruction.

Correct Solution:

1. Bus doesn't need to be aware of them.
2. Everything is local.
3. No ping-pong.
4. No extra hardware (tied to 1)

Correct Frontier Model Response:
The LOAD RESERVE/STORE CONDITIONAL (LdR/StC) pair is preferable over atomic read-test-modify instructions like TEST&SET for the following reasons:

1. Reduced Bus Contention and Cache Invalidation
   TEST&SET performs an atomic read-modify-write operation, which always writes to memory even if the lock acquisition fails. This results in frequent bus traffic and cache invalidations for contending processors, degrading performance.

   LdR/StC avoids unnecessary writes:

   - The StC instruction only writes to memory if the reservation is still valid.
   - Failed StC attempts do not generate writes, reducing bus traffic and cache coherence overhead.

2. No Bus Locking During Critical Section
   TEST&SET requires locking the memory bus for the entire atomic operation, blocking all other memory accesses during execution. This creates a scalability bottleneck in multi-processor systems.

   LdR/StC operates without bus locking:

   - LdR is a standard load instruction, and StC is a conditional store.
   - The hardware tracks reservations passively (e.g., via cache coherence protocols), allowing concurrent memory accesses by other processors between the LdR and StC.
   - This enables better parallelism and scalability for contended synchronization.

The example above illustrates a scenario where a model generates the correct answer to a question in the dataset, and where simple pattern-matching would not suffice for evaluation. The frontier model's response aligns conceptually with the answer key by addressing the same advantages using more verbose phrasing. The model's point about "Reduced Bus Contention and Cache Invalidation" corresponds to "No ping-pong" in the answer key - both describe how LdR/StC prevents unnecessary cache line bouncing between processors by avoiding writes on failed attempts. Similarly, the emphasis on "No Bus Locking During Critical Section" matches "Everything is local" - both highlight that LdR/StC operations can be managed through local reservation tracking without requiring global bus coordination or centralized locking mechanisms.

This example highlights how strict string matching or n-gram-based methods (Papineni et al., 2002; Lin, 2004) may fail to capture semantically correct answers. We therefore elect to use LLM-as-a-judge (Section 4.5) to evaluate FRQ-style questions.

C.7. Case Study #2

This section presents example questions from QUARCH that are similar to the types of questions asked in the memory controller design case study (Section B.7). Please refer to Appendix D.12 for the prompt used for this case study. The prompt asks for the LLM to analyze the memory trace and design parameters to understand their architectural implications on power. The questions in this section require similar reasoning skills.

C.7.1. EXAMPLES FOR ANALYZING MEMORY TRACES

DRAM Row Access Trace Optimization

Context:
Recall from your required reading on Tiered-Latency DRAM that there is a near and far segment, each containing some number of rows. Assume a very simplified memory model where there is just one bank and there are two rows in the near segment and four rows in the far segment. The time to activate and precharge a row is 25ns in the near segment and 50ns in the far segment. The time from start of activation to reading data is 10ns in the near segment and 15ns in the far segment. All other timings are negligible for this problem. Given the following memory request stream, determine the optimal assignment (minimize average latency of requests) of rows in the near and far segment (assume a fixed mapping where rows cannot migrate, a closed-row policy, and the far segment is inclusive).

time 0ns: row 0 read
time 10ns: row 1 read
time 100ns: row 2 read
time 105ns: row 1 read
time 200ns: row 3 read
time 300ns: row 1 read

Question:
What rows would you place in near segment? Hint: draw a timeline.

Correct Solution:
rows 0 and 2. see above

Tiered-Latency DRAM Design Based on Access Pattern

Context:
You would like to understand the configuration of the DRAM subsystem of a computer using reverse engineering techniques. Your current knowledge of the particular DRAM subsystem is limited to the following information:
* The physical memory address is 16 bits.
* The DRAM subsystem consists of a single channel and 4 banks.
* The DRAM is byte-addressable.
* The most-significant 2 bits of the physical memory address determine the bank.
* The DRAM command bus operates at 500 MHz frequency.
* The memory controller issues commands to the DRAM in such a way that no command for servicing a later request is issued before issuing a READ command for the current request, which is the oldest request in the request buffer. For example, if there are requests A and B in the request buffer, where A is the older request and the two requests are to different banks, the memory controller does not issue an ACTIVATE command to the bank that B is going to access before issuing a READ command to the bank that A is accessing.

You realize that you can observe the memory requests that are waiting to be serviced in the request buffer. At a particular point of time, you take the snapshot of the request buffer and you observe the following requests in the request buffer.

Requests in the request buffer (in descending order of request age, where the oldest request is on the top):
Read 0x4C80
Read 0x0140
Read 0x4EC0
Read 0x8000
Read 0xF000
Read 0x803F
Read 0x4E80

At the same time you take the snapshot of the request buffer, you start probing the DRAM command bus. You observe the DRAM command type and the cycle (relative to the first command) at which the command is seen on the DRAM command bus. The following are the DRAM commands you observe on the DRAM bus while the requests above are serviced.

Cycle 0 — PRECHARGE
Cycle 6 — ACTIVATE
Cycle 10 — READ
Cycle 11 — READ
Cycle 21 — PRECHARGE
Cycle 27 — ACTIVATE
Cycle 31 — READ
Cycle 32 — ACTIVATE
Cycle 36 — READ
Cycle 37 — READ
Cycle 38 — READ
Cycle 42 — PRECHARGE
Cycle 48 — ACTIVATE
Cycle 52 — READ

To improve performance, you decide to implement the idea of Tiered-Latency DRAM (TL-DRAM) in the DRAM chip. Assume that a bank consists of a single subarray. With TL-DRAM, an entire bank is divided into a near-segment and far-segment. When accessing a row in the near-segment, the ACTIVATE-to-READ latency reduces by 2 cycles and the ACTIVATE-to-PRECHARGE latency reduces by 5 cycles. When accessing a row in the far-segment, the ACTIVATE-to-READ latency increases by 1 cycle and the ACTIVATE-to-PRECHARGE latency increases by 2 cycles.

Assume that the rows in the near-segment have smaller row ids compared to the rows in the far-segment. In other words, physical memory row addresses 0 through N - 1 are the near-segment rows, and physical memory row addresses N through M - 1 are the far-segment rows.

Question:
If the above DRAM commands are issued 5 cycles faster with TL-DRAM compared to the baseline (the last command is issued in cycle 47), how many rows are in the near-segment? Show your work.

Correct Solution:
59 rows have to be in the near segment.

Explanation. There should 59 rows in the near-segment (rows 0 to 58) since rows until row id 58 need to be accessed with low latency to get 5 cycle reduction. Rows 59 and 192 are in the far-segment, thus latency for accessing them increases slightly.

Here is the new command trace:
Cycle 0 – PRECHARGE - Bank 1
Cycle 6 – ACTIVATE - Bank 1, Row 50, near segment
Cycle 8 – READ - Bank 1
Cycle 9 – READ - Bank 0
Cycle 16 – PRECHARGE - Bank 1
Cycle 22 – ACTIVATE - Bank 1, Row 59, far segment
Cycle 27 – READ - Bank 1
Cycle 28 – ACTIVATE - Bank 2, Row 0
Cycle 30 – READ - Bank 2
Cycle 31 – READ - Bank 3
Cycle 32 – READ - Bank 2
Cycle 39 – PRECHARGE - Bank 1
Cycle 45 – ACTIVATE - Bank 1, Row 58, near segment
Cycle 47 – READ - Bank 1

C.7.2. Examples for Memory Controller Configuration

DRAM Command Design

Context:
You are given a memory system that has four channels, and each channel has two ranks of DRAM chips. A separate memory controller controls each memory channel. Each rank of DRAM contains eight banks. A bank contains R rows. Each row in one bank is 8KB. The minimum retention time among all DRAM rows in the system is 64 ms. In order to ensure that no data is lost, every DRAM row is refreshed once per 64 ms. Refresh of each row is initiated by a command from the memory controller. The command refreshes only the specified row. The command occupies the command bus on the associated memory channel for 5 ns and the associated bank for 40 ns.

We define refresh utilization of a resource (such as a bus or a memory bank) as the fraction of total time for which a resource is occupied by a refresh command.

Question:
How can you reduce the command bus utilization due to refreshes? You cannot change the refresh rate when answering this question.

Correct Solution:
Have each command be responsible for multiple row refreshes.

DRAM Hierarchy Configuration

Context:
Recall from your required reading on Tiered-Latency DRAM that there is a near and far segment, each containing some number of rows. Assume a very simplified memory model where there is just one bank and there are two rows in the near segment and four rows in the far segment. The time to activate and precharge a row is 25ns in the near segment and 50ns in the far segment. The time from start of activation to reading data is 10ns in the near segment and 15ns in the far segment. All other timings are negligible for this problem. Given the following memory request stream, determine the optimal assignment (minimize average latency of requests) of rows in the near and far segment (assume a fixed mapping where rows cannot migrate, a closed-row policy, and the far segment is inclusive).

time 0ns: row 0 read
time 10ns: row 1 read
time 100ns: row 2 read
time 105ns: row 1 read
time 200ns: row 3 read
time 300ns: row 1 read

Question:
Assume now that the mapping is dynamic. What are the tradeoffs of an exclusive design vs. an inclusive design? Name one advantage and one disadvantage for each.

Correct Solution:
Exclusive requires swapping, but can use nearly full capacity of DRAM. Inclusive, the opposite.

DRAM Sizing Optimization

Context:
You are given a memory system that has four channels, and each channel has two ranks of DRAM chips. A separate memory controller controls each memory channel. Each rank of DRAM contains eight banks. A bank contains R rows. Each row in one bank is 8KB. The minimum retention time among all DRAM rows in the system is 64 ms. In order to ensure that no data is lost, every DRAM row is refreshed once per 64 ms. Refresh of each row is initiated by a command from the memory controller. The command refreshes only the specified row. The command occupies the command bus on the associated memory channel for 5 ns and the associated bank for 40 ns.

We define refresh utilization of a resource (such as a bus or a memory bank) as the fraction of total time for which a resource is occupied by a refresh command.

Question:
Only changing the number of rows per bank, find the maximum number of rows per bank for which either the bank utilization or the command bus utilization reaches 100%.

Correct Solution:
Because the command bus utilization will reach 100% before the bank utilization, we will look at how changing the number of rows will affect the command bus utilization
R*5ns*2ranks*8banks/64ms = 1
R = 800,000
We find that with 800k rows per bank, the command bus utilization reaches 100%.

D. Prompt Templates

D.1. LLM Prompt for MCQs

MCQ Prompt

You are an expert computer architect solving multiple choice questions. Please read the following question and select the best answer from the choices provided.

Question:
{question}

Choices:
{choices}

Please conclude your response with a JSON object containing your final answer. The JSON object must match this schema: {"final_answer": "⟨ A, B, C, or D⟩"}.

This prompt is designed for multiple-choice evaluation, where correctness can be measured directly. The JSON-constrained output ensures answers are machine-readable and easy to score at scale.

D.2. LLM Prompt for FRQs

FRQ Prompt

You are an expert computer architect taking an exam. You will be provided with a question and its context. Your task is to provide a clear, accurate, and well-reasoned answer to the question.

Please provide your answer in a structured format that clearly addresses the question. If the question involves calculations, show your work step-by-step. If it involves diagrams or tables, describe them clearly.

Remember to:

1. Read the question carefully and understand what is being asked
2. Use the provided context to inform your answer
3. Show your reasoning and work where appropriate
4. Be precise and accurate in your response
5. If you're unsure about something, acknowledge the uncertainty

Question Context:
{context}
{context_images_placeholder}

Question:
{question}

Please provide your answer:

This prompt is used to simulate a "student" LLM, where models act as exam-takers solving architecture questions. The emphasis is on structured, step-by-step reasoning, accuracy, and clarity, mirroring how a human student would respond to technical exam questions.

D.3. LLM Prompt for LLM-as-a-Judge on FRQ Responses

LLM-as-a-Judge Prompt

You are an expert computer architect acting as an exam grader to evaluate the quality of an answer to a computer architecture question. You will be provided with:

1. The original question and context
2. The correct solution
3. A student's answer to the question

Your task is to carefully evaluate whether the student's answer is correct, partially correct, or incorrect by comparing it to the provided solution.
Evaluation criteria:

- CORRECT: The answer is accurate, complete, and demonstrates proper understanding
- PARTIALLY-CORRECT: The answer shows some understanding but has significant errors or is incomplete
- INCORRECT: The answer is fundamentally wrong or shows major misunderstandings

Consider:

- Mathematical accuracy
- Conceptual understanding
- Completeness of the response solely in relation to the question being asked
- Logical reasoning
- Whether the answer addresses what was actually asked

Be fair but rigorous in your evaluation. If you are unsure, err on the side of being more critical.

Question Context:
{context}
{context_images_placeholder}

Question:
{question}

Correct Solution:
{solution}
{solution_images_placeholder}

Student's Answer:
{student_answer}

Please evaluate the student's answer and provide your reasoning. At the end of your response, write exactly one of the following words in all caps on a new line: CORRECT, PARTIALLY-CORRECT, or INCORRECT. If you do not end your response with a new line with exactly one of these options, you will not be paid for your work.

This prompt is used to evaluate model outputs under the "LLM-as-a-Judge" paradigm. Here, the model acts as a grader, comparing student answers to reference solutions and deciding between CORRECT, PARTIALLY-CORRECT, or INCORRECT. It enables scalable evaluation of free-response questions while preserving rigor and consistency.

D.4. LLM Prompt for Skills Classification

Skill Classification Prompt

You are a computer architecture expert who is a professional at categorizing exam questions based on the cognitive skill they are testing of a computer architect. Your task is to classify one question at a time into exactly one of the following four categories:
Recall: The question asks for a fact, definition, or direct retrieval of knowledge. Answering these questions should typically not require multi-step reasoning and can be directly answered in a single step.

Analyze: The question requires deducing, inferring, calculating, or interpreting information based on data or a specific scenario. Answering these questions typically requires some level of multi-step reasoning but does not require invention or innovating upon an existing solution.

Design: The question asks you to propose, invent, suggest, or improve a method, system component, or policy. It requires proposing new or improved solutions.

Implement: The question requires constructing, coding, or developing a full solution or system based on explicit requirements or specifications. These typically involve providing detailed instructions or actual code (i.e., programmable/executable artifacts).

Sometimes "Analyze" and "Design" can be confused if the question is open-ended. If the main effort is proposing or inventing, choose "Design"; if it's interpreting specific data or information, choose "Analyze".

Additionally, sometimes "Design" and "Implementation" can also be confused. If the answer involves some sort of programmatic implementation or executable artifact then choose "Implement"; if the answer is at a higher-level of abstraction than this, it is likely to be "Design".

Your output **MUST** be **ONLY** one word, chosen from the following list:
Recall, Analyze, Design, Implement

Find an example below:
Recall: A ____ cache allows any block of main memory to be placed in any line, eliminating conflict misses but requiring complex associative lookup hardware.
Analyze: A fully associative cache has 4 lines and uses an LRU policy. The following sequence of memory references occurs . . . What is the overall hit ratio?
Design: Describe a cache replacement policy that would improve the performance of the hybrid memory system more than it would DRAM.
Implement: Create a cache controller that interfaces with both a processor pipeline and a DRAM chip, following the provided Verilog port and signal specifications.

Now, classify this question:
{question}

We use this prompt to consistently label each exam or benchmark question with the specific cognitive skill it targets. Our rationale in writing it was twofold: (i) provide clear, operational definitions of *Recall*, *Analyze*, *Design*, and *Implement* that are grounded in how architects approach problem solving, and (ii) minimize ambiguity by explicitly addressing common confusions between neighboring categories (e.g., Analyze vs. Design, or Design vs. Implement). This ensures that classification is reliable across reviewers and that the benchmark's skill taxonomy aligns with real-world architectural workflows.

D.5. LLM Prompt for Architecture Topic Classification

To determine the architecture topic distribution of QUARCH, we employed a two-stage classification process that combines embedding-based similarity search with LLM reasoning for scalable categorization. In the first stage, the top 3 most relevant topic candidates are identified by embedding each question using OpenAI's text-embedding-3-large model and comparing against pre-computed taxonomy topic embeddings via cosine similarity. In the second stage, GPT-4O is used to make the final topic selections from these 3 candidates, providing both best and second-best classifications along with justifications.

This hybrid approach effectively balances accuracy with scalability by leveraging the computational efficiency of embedding-

based similarity search for initial filtering, while utilizing LLM reasoning capabilities for nuanced final classification decisions. Below is the prompt used for LLM categorization:

Architecture Topic Classification Prompt

You are a computer architecture researcher and expert. You have been asked to categorize the following question into a subfield of computer architecture. Three options have been provided and you must select the top two.

Question: {question}

Categories:
1: {taxonomy_terms[0]}
2: {taxonomy_terms[1]}
3: {taxonomy_terms[2]}

Please provide the exact names of the two categories that you feel best fit this question. First select the best match of the three options and then choose the second best category that matches. You have been told **you have to pick no matter what from the options** and your response should be in the format without any additional text or explanation:

[
{
"best_selection": "[BEST CATEGORY HERE]",
"justification": "[JUSTIFICATION #1 HERE]"
},
{
"second_best_selection": "[SECOND BEST CATEGORY HERE]",
"justification": "[JUSTIFICATION #2 HERE]"
}
]

D.6. LLM Prompt for Synthetic MCQ Generation

MCQ Generation Prompt

You are a computer architecture professor and expert researcher. You have been provided with the following excerpt about computer architecture:
"{excerpt}"

You have been asked to create one difficult, paraphrased cloze-style format multiple choice question based on this excerpt to test senior computer architects.

The questions will be used for creating an interview test for senior computer architects, and they will not get to read the excerpt for context, so any questions you create should not refer to anything specifically in the excerpt that would make the question unanswerable in the excerpts's absence.

The questions must be precise and clear so they can be answered *definitively*, so avoid using qualifying adjectives or adverbs that would make the answer ambiguous or depend on the context; make sure there is only one correct answer to the question.

Quote *word for word* the sentence(s) of the excerpt context that are comprehensive and self-contained and could be read to justify and support each answer to each question. Your goal is to create good **conceptual** questions that test **conceptual** knowledge from the excerpt.

Here is an example of a good cloze question:
____ is the typical penalty incurred for a branch mispredict.
A) 100 us
B) 5 ms
C) 5 ns
D) 100 ms
Answer: C

Provide your response in this *exact* format with *zero additional characters for formatting* before or after the opening and closing brackets:
[
{
“question”: “[CLOSE QUESTION HERE]”,
“option A”: “[OPTION A HERE]”,
“option B”: “[OPTION B HERE]”,
“option C”: “[OPTION C HERE]”,
“option D”: “[OPTION D HERE]”,
“answer”: “{specified_answer_choice[‘Cloze’]}”,
“context”: “[JUSTIFICATION HERE AS CONTEXT]”,
“type”: “Cloze”
}
]

D.7. LLM Prompts for Filtering Synthetic MCQs

The following prompts were used to filter out poor MCQs that were synthetically generated. QAs that passed all of these filters were then finally verified by humans for quality and correctness.

MCQ Filtering Prompt #1

You are a computer architecture expert.

You have been asked the following question:
{question}

Here are the options:
{options}

As a computer architecture expert, would you need additional context to correctly answer the question?

Please answer with one word: “YES” or “NO”.

Then provide one short sentence to justify your answer reasoning. Return your response in this exact format with zero other characters for formatting before or after:
{
“answer”: “[ANSWER HERE: YES/NO]”,
“justification”: “[JUSTIFICATION HERE]”
}

MCQ Filtering Prompt #2

You are a computer architecture expert.

You have been asked the following question:
{question}

Here are the options:
{options}

As a computer architecture expert, of the provided options is there *only one* answer that is correct?

Please answer with one word: “YES” or “NO”.

Then provide one short sentence to justify your answer reasoning. Return your response in this exact format with zero other characters for formatting before or after:
{
“answer”: “[ANSWER HERE: YES/NO]”,
“justification”: “[JUSTIFICATION HERE]”
}

MCQ Filtering Prompt #3

You are a new graduate student reading about computer architecture to learn the subject.

You have been given the following quote from a computer architecture excerpt to read for context:
{context}

You have now been asked the following question:
{question}

Here are the options:
{options}

Does the provided context you read sufficiently help you answer the question correctly?

Please answer with one word: “YES” or “NO”.

Then provide one short sentence to justify why you answered “YES” or “NO”. Return your response in this exact format with zero other characters for formatting before or after:
{
“answer”: “[ANSWER HERE: YES/NO]”,
“justification”: “[JUSTIFICATION HERE]”
}

MCQ Filtering Prompt #4

You are a computer architecture expert.
You have been given the following quote from a computer architecture excerpt for context:
{context}

You have now been asked the following question:
{question}

Here are the options:
{options}

Choose the best correct option and provide your answer justification. Return your response in this exact format with zero other characters for formatting before or after:
{
"answer": "[ANSWER HERE: LETTER OF OPTION]",
"justification": "[JUSTIFICATION HERE]"
}

D.8. LLM Prompt for Text Extraction of Exam QAs

This is the prompt used to extract the text for questions in our crowdsourced exams. It ensures that problem statements, sub-questions, and solutions are consistently structured into JSON, while ignoring irrelevant formatting such as point values or images. By enforcing this schema, we can standardize raw exam PDFs into machine-readable data suitable for validation, benchmarking, and downstream analysis.

Exam Question Text Extraction Prompt

You are a language model assisting with the digitization of academic exam content. The input is a PDF file containing one problem of a computer architecture assessment. If part of another problem is included, ignore it and only focus on {filename}.

The problem may include any combination of the following:
A context paragraph, or just a short statement (e.g., "Convert the number 42 to binary")
One or more sub-questions, or be a single standalone question
Context for sub-questions separate from the sub-question and separate from the original problem context
Multiple questions within a subquestion
Point value associations for the problem or subproblems, including extra credit points
Solutions, either typed or handwritten
Tables, diagrams, circuit schematics, or block diagrams

Your task is to identify and separate each exam problem into the listed components, including context, sub-questions, and solutions. At times, a subquestion can have nested subparts. Ignore any point values for any problem, question, or sub-question. Ignore any images, charts, or figures and do not attempt to extract text from them.

If a provided image is not part of the problem in the pdf file and instead is part of another problem(s), omit it from the dictionary. Format your response so that it can be exported into a JSON file using the template below. If the particular exam question lacks any of the listed components, omit them from the template.

Template for a problem which is split up into sub-problems:

```
{
    "problem": "1",
    "problem_context": <Insert any introductory paragraph or description exactly as
        it appears. If there is no context, don't include this header>,
    "subproblems": [
    {
        "subproblem": "a" (Copy the part letter/number exactly as it appears on the
            exam),
```

```
        "subproblem_context": <Insert any introductory paragraph or description
            exactly as it appears. If there is no subproblem context or if the
            question is the only part of the subproblem, don't include this header.
            Replace all double quotes " here with escaped double quotes /">,
        "subproblem_question": <Insert the full question of the subproblem, exactly
            as it appears in the original. Replace all double quotes " here with
            escaped double quotes /">,
        "subproblem_solution": <Insert the full solution of the subproblem, exactly
            as shown in the original. Replace all double quotes " here with escaped
            double quotes /">
    },
    ...repeat as needed for additional subproblems within this problem
    ],
}
```

Template for a problem which is standalone and has no sub-problems:

```
{
    "problem": "1",
    "problem_context": <Insert any introductory paragraph or description exactly as
        it appears. If there is no context, don't include this header. Replace all
        double quotes " here with escaped double quotes /">,
    "problem_question": <Insert the full question of the problem, exactly as it
        appears in the original. Replace all double quotes " here with escaped
        double quotes /">,
    "problem_solution": <Insert the full solution of the problem, exactly as shown
        in the original. Replace all double quotes " here with escaped double quotes
         /">
}
```

If any double quotes (") within strings appear within your JSON, you must replace them with escaped double quotes (/"). Return only valid JSON.

D.9. LLM Prompt for Image Extraction of Exam QAs

This is the prompt used to associate figures with their correct roles in the digitized exam problems. By structuring images into categories such as problem context, subprobem context, problem solution, and subproblem solution, we can align visual information with text-based question content. This ensures that diagrams, tables, and schematics are consistently linked to their intended question or solution, enabling precise and reproducible dataset construction.

Exam Question Image Extraction Prompt

You are a language model assisting with the digitization of academic exam content in computer architecture.

Input:
You are provided with:
1) A PDF file containing one problem of an exam. If part of another problem is included, ignore this and only focus on the current pdf file.
2) A .json-styled txt file containing the problem's extracted text. It may contain some or all of the following empty-list fields: "`problem_context_figures`", "`subproblem_context_figures`", "`subproblem_solution_figures`", and "`problem_solution_figures`".
3) PNG images containing tables, diagrams, circuit schematics, or block diagrams that may or may not pertain to this problem. The names of the images provided are as follows, in order: {images}.

The given problem may be a standalone problem or consist of multiple sub-problems. Each problem or sub-problem may contain:
– Main context figures which are necessary to understanding the main problem, or ALL of the sub-problems.
– Sub-problem-specific figures that are separate from both the main problem context and the other sub-problems.

– Solutions to the main problem, which may be typed or handwritten.
– Solutions to a certain sub-problem, which may be typed or handwritten.

Your task:
Match each image file name (table, diagram, circuit schematic, or block diagram) to its correct association in the exam. Each image should have one of the four possible associations:
– The main problem question ("`problem_context_figures`")
– The sub-problem question ("`subproblem_context_figures`")
– The main problem solution ("`problem_solution_figures`")
– The sub-problem solution ("`subproblem_solution_figures`")

At times, the context or question in the main problem/sub-problem question/solution will include phrases (such as "The table below" or "The following diagram") that indicate a visual image falls under that category.

Your output should be a modified version of the given JSON, but with each of the figure fields populated with lists of the relevant image names (files ending in .png, .jpg, etc.), for example:
"`problem_context_figures`": ["`image_name.png`", "`image_name_2.png`"]

No content, descriptions, or recreations of the image should be included in the output; only the file name should be included. If a provided image is not part of the current PDF file and instead is part of other problem(s), omit it. If the provided JSON includes image file names that are not present in the images provided, omit those too.

Be as precise as possible in your associations. Only include the dictionary; do not include reasoning. If none of the images given pertain to the problem, just output the original JSON given.

D.10. LLM Prompt for Verification of Extracted Exam QAs

This prompt is used to audit the parser's extracted question–answer pairs against the original exam source. It enforces strict, itemized criteria for textual fidelity, self-contained context, correct image/table extraction, and proper figure categorization, and it standardizes the verification output by toggling `passed_llm_verification` and adding concise reasoning where failures are detected.

Exam Question Verification

You are a reader tasked with verifying the accuracy of an automated document parser. A computer architecture exam PDF has been fed through the parser to produce standalone question–answer pairs in a certain JSON format, and you must compare the parsed questions to the original questions in the PDF.

Input: You are given (1) the original PDF of {problem_name} of the exam which may be broken down into subproblems, (2) the JSON dictionary produced by the parser, and (3) images that the parser has deemed are associated with the question and/or its subproblems. The names of the images provided are as follows, in order: {images}. Your task is to determine if the parser has correctly extracted the text and images while staying true to the original PDF.

A problem is correct only if all of the following are satisfied:
a) The extracted problem text is nearly identical, word-for-word, to the original PDF's text (special characters/math symbols may appear as unicode-escaped or equivalent).
b) The problem's "`question`" and "`solution`" fields are both populated. Exception: if the original solution is purely an image, that image must be correctly associated in "`solution_figures`".
c) The problem is standalone: the "`context`" and "`context_figures`" provide all information needed, even if the original referenced prior problems.
d) No part of the solution is revealed in "`context`" or "`context_figures`".

e) All images are extracted/cropped correctly and categorized correctly as context vs. solution figures.
f) All tables are extracted correctly. If parsed as text instead of image, the table must be recreatable from the extracted text and usable to answer the question.
g) For fill-in-the-blank or fill-in-the-chart, the blank version must be provided in the question/context or "`context_figures`".

Output format:
Return a modified version of the given JSON dictionary. Set each "`passed_llm_verification`" field to `true` or `false` (unquoted). For every item marked `false`, add a "`reasoning`" field explaining which conditions failed. Return exactly one dictionary and nothing else.
Conservatism: Err on the side of false. Avoid false positives; false negatives are acceptable.

D.11. LLM Prompts for Case Study #1

The following prompt was used to filter out QUARCH FRQs related to memory subsystems for our case study in Section 4.4, using CLAUDE-3.7-SONNET-THINKING:

LLM-assisted Filtering of Memory Subsystem FRQs

You are an expert computer-architecture instructor. You will be given a question, its context, and the provided solution. You need to decide whether the question directly tests the necessary skills and knowledge for designing a memory subsystem with caches for a matrix multiplication workload. Full walkthroughs of selected questions will be used to teach an experienced engineer the necessary reasoning steps for optimizing memory performance for matrix multiplication. {`question`}{`solution`}

The following three user instruction prompts were used to produce SFT samples for data distillation. For each of the 45 QUARCH FRQs filtered using the above prompt, the teacher model (GPT-5.2) targeted three student types via separate roles:

Teacher Prompt for Student Type #1: Junior Architect in Industry

You are a senior computer architect that is an expert in memory subsystems. You are training a less-experienced computer architect to take over your role as a cache designer. They have brought you the following question and solution from a graduate academic computer architecture exam, and asked you to explain the underlying principles an expert needs to know about in order to determine why this solution is the correct one, including nuances that they may be unaware of even if they answered correctly and completely. In particular, you should precisely convey the architectural principles and considerations relevant specifically to this exam question, and briefly touch on any counterexamples or connections that will reinforce their expertise in cache and memory design. Reason deeply about this exam question and solution, and consider what you can teach the architect. Keep your final didactic response to under 5 paragraphs, and keep in mind they understand the fundamentals of memory subsystems and caches at a technical level already. {`question`}{`solution`}

Teacher Prompt for Student Type #2: Computer Architecture Student

You are a professor in computer architecture that is an expert in memory subsystems. A student has brought you a question and answer from a graduate academic computer architecture exam, and asked you to explain the underlying principles behind why this solution is the correct one. In particular, when teaching them how to solve this problem you should carefully and precisely bolster their ability to solve a wide array of memory subsystem problems, and reinforce their skills in cache and memory design topics. Reason deeply about this exam question and solution, and keep your final didactic response to under 5 paragraphs (do not include basic definitions and cursory explanations, as they have taken your introductory computer architecture course already). {`question`}{`solution`}

Teacher Prompt for Student Type #3: Novice

You are a professor in computer architecture that is an expert in memory subsystems. A talented university student has brought you a question and answer from a graduate academic computer architecture exam, and asked you to explain how to arrive at the solution. The student has taken an introductory computer architecture class many years ago and remembers term definitions and can provide cursory explanations of concepts, but they don't know how to begin to solve this particular problem and require a careful walkthrough. In particular, when teaching them how to solve this problem you should bolster their ability to solve other questions of this type, topic, and difficulty, and heavily reinforce their skills in cache and memory design topics. Every word counts and you must be incredibly precise and technically dense in your response to achieve these goals. This is the culmination of your life's work as a subject matter expert and teacher. Reason deeply about this exam question and solution, and keep your final didactic response to under 5 paragraphs. {question}{solution}

D.12. LLM Prompt for Case Study #2

The following prompt is used for the DRAM controller case study (Appendix Section B.7). It tasks an LLM with answering 3 questions and then proposing a new memory controller design configuration. current_iteration is the i-th round of the 40 evaluation budget provided to the LLM. Thirty lines of the beginning, middle, and end of the workload's memory trace are sampled to add to the prompt. Some example configurations proposed by an LLM during a trial in this case study are provided for illustration in the prompt below. Note that ellipses are used for brevity but not actually a part of the prompt.

Memory Controller Design Configuration Task

You are a computer architecture expert specializing in DRAM memory controller design. You have been tasked with tuning the design of a custom DRAM memory controller to optimize for **POWER** (lower is better).

=== SEARCH BUDGET ===
You are on iteration {current_iteration} of 40.
Consider this when deciding your strategy:
- Early iterations: Consider exploring diverse configurations to understand the design space.
- Later iterations: Consider exploiting promising regions by making smaller refinements to the best configurations.

=== DRAM CONTROLLER DESIGN PARAMETERS ===
You control 10 DISCRETE design parameters, each represented as an INTEGER INDEX:
0: PagePolicy, Options: ["Open", "OpenAdaptive", "Closed", "ClosedAdaptive"]
1: Scheduler, Options: ["Fifo", "FrFcfsGrp", "FrFcfs"]
2: SchedulerBuffer, Options: ["Bankwise", "ReadWrite", "Shared"]
3: RequestBufferSize, Options: [1, 2, 4, 8, 16, 32, 64, 128]
4: RespQueue, Options: ["Fifo", "Reorder"]
5: RefreshPolicy, Options: ["NoRefresh", "AllBank"]
6: RefreshMaxPostponed, Options: [1,2,4,8]
7: RefreshMaxPulledin, Options: [1,2,4,8]
8: Arbiter, Options: ["Simple", "Fifo", "Reorder"]
9: MaxActiveTransactions, Options: [1, 2, 4, 8, 16, 32, 64, 128]

=== CONFIGURATIONS TRIED SO FAR ===
Here are the top 10 best configurations you've found so far, ranked by Power (W) (best first):
#1: [1.0, 2.0, 2.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0], Energy=0.000194 J, Power=1.697670 W, Latency=0.000114290 s
#2: [1.0, 2.0, 2.0, 2.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0], Energy=0.000178 J, Power=2.092590 W, Latency=0.085218750 s
#3: [1.0, 2.0, 2.0, 3.0, 1.0, 0.0, 0.0, 0.0, 0.0, 2.0], Energy=0.000171 J, Power=2.408910 W, Latency=0.000070795 s
...

Here are the last 10 configurations you've tried:
#1: [1.0, 2.0, 2.0, 3.0, 1.0, 0.0, 0.0, 0.0, 0.0, 2.0]
#2: [1.0, 2.0, 2.0, 2.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0]
#3: [1.0, 2.0, 2.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
...

IMPORTANT: You CANNOT propose a configuration that exactly matches any of the Top 10 or Last 10 configs you've tried listed above as proposing a duplicate configuration wastes an evaluation.

=== MEMORY TRACE SAMPLE ===
Below is a sample of the workload's memory trace (format: "timestamp: operation address"):
— Beginning (lines 0-29 of 10000) —
0: read 0x0
1: read 0x40
2: read 0x80
3: read 0xC0
4: read 0x100
...

— Middle (lines 4985-5014 of 10000) —
4985: read 0x190BBCC0
4986: read 0x26EEB740
4987: read 0x29CEE6C0
4988: read 0x3771780
4989: read 0x277DC1C0
...

— End (lines 9970-9999 of 10000) —
9970: read 0x4A880
9971: read 0x4A8C0
9972: read 0x4A900
9973: read 0x4A940
9974: read 0x4A980
...

=== YOUR TASK ===
Before proposing a new configuration, use your understanding of DRAM memory controller architecture to reason through the following to make an informed choice:

1. **Memory Access Pattern Analysis**: Looking at the memory trace sample above, what are key characteristics of this workload that you should consider when tuning the DRAM memory controller?

2. **Parameter Impact Analysis**: Based on the trace sample and broader optimization target (POWER) of yours, which parameters are most critical to tune and why?

3. **Learning from History**: Looking at previous configurations and their metrics, what patterns do you observe? Which parameter changes led to improvements?

Based on your analysis from the THREE QUESTIONS above and the remaining search budget, propose one NEW configuration that you expect will MINIMIZE POWER.

IMPORTANT OUTPUT FORMAT:
- First, provide your reasoning in a <reasoning>block
- Then, output ONLY a JSON list of exactly 10 integers (the indices 0..7 or 0..3 etc) on its own line
- Do NOT include any extra text beyond the <reasoning>and JSON list of integer indices.

Example output format (enclosed in markdown code block):

```
<reasoning>
My analysis shows that ... therefore, I propose the following configuration:
</reasoning>
indices=[2, 0, 1, 3, 1, 0, 2, 1, 2, 4]
```

which in this example would correspond to the following DRAM controller design configuration:
- PagePolicy: “Closed” (index 2)
- Scheduler: “Fifo” (index 0)
- SchedulerBuffer: “ReadWrite” (index 1)
- RequestBufferSize: 8 (index 3)
- RespQueue: “Reorder” (index 1)
- RefreshPolicy: “NoRefresh” (index 0)
- RefreshMaxPostponed: 4 (index 2)
- RefreshMaxPulledin: 2 (index 1)
- Arbiter: “Reorder” (index 2)
- MaxActiveTransactions: 16 (index 4)