

QUARCH: A BENCHMARK FOR EVALUATING LLM REASONING IN COMPUTER ARCHITECTURE

Anonymous authors

Paper under double-blind review

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 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 72%) on these advanced questions, highlighting persistent gaps in architectural reasoning across analysis, design, and implementation QAs. By holistically assessing fundamental skills, QUARCH provides a foundation for building and measuring LLM capabilities that can accelerate innovation in computing systems.

1 INTRODUCTION

Benchmarks that elicit reasoning are among the most impactful large language model (LLM) evaluations today since they require critical thinking that goes beyond surface-level knowledge and pattern matching. As a result, state-of-the-art (SOTA) progress is tracked on benchmark suites requiring multi-step reasoning, where models with explicit test-time deliberation (i.e., “thinking” variants) consistently climb leaderboards. Widely adopted datasets such as GSM8K (Cobbe et al., 2021), AIME (Mislav Balunović, 2025), SWE-bench (Jimenez et al., 2024), GPQA (Rein et al., 2023), and MMLU-Pro (Wang et al., 2024a) probe this structured reasoning to 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 any LLM benchmarks to date.

Existing benchmarks in computing systems target engineering tasks for software or chip implementation such as code generation (Jimenez et al., 2023; 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., 2025). 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.

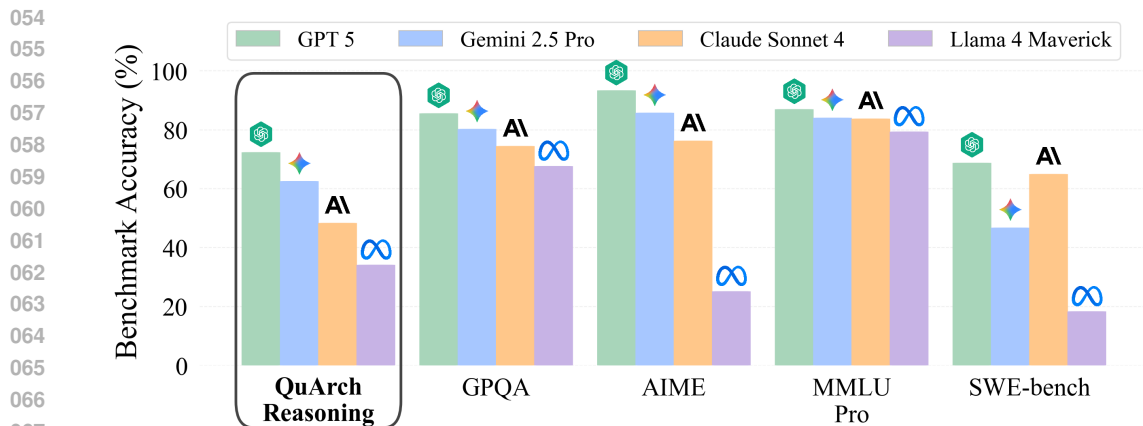


Figure 1: Reported results (Vals AI, Inc., 2025) for different models across QUARCH and multiple other SOTA benchmarks to date.

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 broad domain knowledge and skill set that architects possess by constructing the benchmark around four foundational competencies: Recall, Analyze, Design, and Implement. Existing computing systems benchmarks primarily target technical implementation skills (Table 1), but all four competencies—recalling foundational principles, analyzing workloads and constraints, designing systems that balance objectives, and implementing solutions via code—are complementary and critical for effective architecture design. Although we focus on computer architecture, these skills are broadly applicable to many systems tasks. Our framework enables systematic evaluation that holistically measures an agent’s ability so that model progress can be measured with fine-grained skills and compared over time.

In summary, our work makes the following contributions: **1** QUARCH 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. **2** To promote holistic evaluation of AI agents 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. **3** 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. **4** 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 85.35%, which is comparable to human-human grading agreement rates of 90.7% on the QUARCH benchmark.

2 QUARCH

2.1 TOWARDS AGENTIC SYSTEMS DESIGN: BUILDING AN AI ARCHITECT

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

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., 2023) | Software Eng. | 7 | 3 | 7 | 7 | 2294 |
| SWE-bench Verified (OpenAI, 2024) | Software Eng. | 7 | 3 | 7 | 3 | 500 |
| SWE-Perf (He et al., 2025) | Performance Eng. | 7 | 3 | 7 | 7 | 140 |
| KernelBench (Ouyang et al., 2025) | Performance Eng. | 7 | 3 | 7 | 7 | 250 |
| CodeMMLU (Nguyen et al., 2025) | Code Reasoning | 3 | 7 | 7 | 7 | 19912 |
| CRUXEval (Gu et al., 2024) | Code Reasoning | 3 | 3 | 7 | 7 | 800 |
| QUARCH (This Work) | Architecture | 3 | 3 | 3 | 3 | 2671 |
| SLDB (Alvanaki et al., 2025) | System Design | 7 | 3 | 7 | 3 | 10 |
| CreativEval (DeLorenzo et al., 2024b) | HW Design | 7 | 3 | 7 | 7 | 120 |
| VerilogEval (Liu et al., 2023b) | RTL Generation | 7 | 3 | 7 | 7 | 156 |
| CVDP (Pinckney et al., 2025b) | RTL Generation | 3 | 3 | 7 | 3 | 783 |
| MG-Verilog (Zhang et al., 2024) | RTL Generation | 7 | 3 | 7 | 7 | 11000 |
| EDA Corpus (Wu et al., 2024a) | EDA Tooling | 3 | 3 | 7 | 3 | 1533 |
| FIXME (Wan et al., 2025) | Verification | 7 | 3 | 7 | 7 | 180 |
| ChiPBench (Wang et al., 2024b) | Layout | 7 | 3 | 7 | 7 | 20 |

architects (and systems engineers more broadly) require. Figure 2 illustrates these skills: within a single problem scenario, we exemplify how different styles of QAs exercise different skills, from recall of fundamental domain knowledge to more advanced analysis, design, and implementation.

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 critically important for deeper understanding. “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 often can require iterating over architectural block diagrams and system specifications. “Suggest a dynamic branch predictor that integrates into the 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.

Knowledge Breadth Requirements. Computer architecture contains a multitude of specialized areas. Historically, architects focused on microprocessor design but 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; Esmailzadeh et al., 2011). Consequently, this elevated the importance of understanding memory systems, interconnects, and system-level methodology (Sangiovanni-Vincentelli, 2007; Carloni, 2015) to first-class concerns. Effective architecture and system design requires understanding relationships across these areas and reasoning about how they interact. For example, a processor aggressively optimized without considering the connected memory subsystem will exhibit more performance bottlenecks than if the two were optimized together. Thus, a benchmark should capture topic breadth to properly assess architectural knowledge.

162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215

Figure 2: QUARCH QA Skills Framework. The benchmark evaluates four complementary competencies: 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. Together, these competencies form the building blocks required for systems design in computer architecture. Balanced mastery of skills will enable more effective agents and workflows across the computing stack.

2.2 BENCHMARK CONSTRUCTION

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 three-pronged strategy that combines synthetic data generation, academic exams, and expert crowdsourcing and competitions (Fig. 3). Every QA is reviewed and expert-validated by doctoral students with graduate-level training in computer architecture before inclusion in QUARCH.

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 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., 2025), 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 (Ap-

216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269

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.

pendix B.1). 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 QUARCH’s 2,671 QA pairs 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).

Figure 4: Distribution of topics in QUARCH.

Skills Coverage. Figure 5 characterizes QAs by the skills in Sec. 2.1, derived from LLM labeling (Appendix D.4) with examples for each given in Appendix C.1. In particular, QUARCH targets advanced reasoning by providing nearly 1000 analysis QAs and 100 design & implementation

270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323

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

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

QAs. This disparity in number 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 term recall-focused QAs QUARCH-RECALL and higher-order skill QAs (e.g., analyze, design, implement) QUARCH-REASONING.

Question Format and Input Modalities. In line with prior QA and code-reasoning benchmarks (Rein et al., 2023; 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, routine 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 data flow 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.

3 EXPERIMENTAL SETUP

Models. We evaluate ten frontier models from Alibaba Cloud, Anthropic, DeepSeek, Google, Meta, Mistral, and OpenAI on QUARCH across the four skills (Recall, Analyze, Design, Implement) presented in Section 2.1. Evaluation results for an additional 16 models are reported in Appendix B.4.

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.

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 **rst**, **second**, and **third** best performing models.

| Model | QUARCH-RECALL | QUARCH-REASONING | |
|----------------------------|---------------|------------------|-------|
| Multimodal Models | | | |
| GPT-5 | 89.0 | 72.4 | -16.5 |
| GPT-5 (Non-Reasoning) | 86.3 | 49.0 | -37.3 |
| Gemini 2.5 Pro | 87.4 | 62.9 | -24.5 |
| Gemini 2.5 Flash | 83.4 | 56.8 | -26.6 |
| Claude Sonnet 4 | 85.5 | 48.4 | -37.0 |
| Claude 3.7 Sonnet Thinking | 85.8 | 52.1 | -33.7 |
| Llama 4 Maverick | 85.3 | 34.2 | -51.1 |
| Mistral Medium 3.1 | 84.5 | 34.1 | -50.4 |
| Text-Only Models | | | |
| GPT-OSS 120B | 84.2 | 64.7 | -19.5 |
| DeepSeek R1 | 86.9 | 56.1 | -30.7 |

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 parameters. In all evaluations, we generate n = 3 samples per question.

4 EVALUATION & ANALYSIS

4.1 MODEL PERFORMANCE

Table 2 reports headline accuracy of 10 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). 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. Frontier models have largely mastered recall, but fall short on advanced skills. Recall accuracy ranges between 83%-89%, suggesting architectural knowledge is present. However, analyze, design, and implement skills are lower than recall by 32.1%, 39.2%, and 38.2% 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, design skills exhibit the widest performance gaps, ranging between 18-87%. 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-thinking with its non-reasoning variant supports this. GPT-5 outperforms by 23%, 34%, and 30% on analyze, design, and implement QAs respectively, compared to a much smaller 3% lift on recall QAs. Moreover, GPT-5 stands as an extreme outlier on design QAs at 89%, suggesting its training corpora and test-time reasoning budget better capture the skills needed to select and justify design decisions under constraints compared to other frontier models.

Table 3: Per-generation accuracy (%) by QuArch Skill. Best performing models in each category highlighted *rst*, *second*, and *third*.

| Model | QUARCH-REASONING | | | |
|----------------------------|------------------|---------|--------|-----------|
| | Recall | Analyze | Design | Implement |
| Multimodal Models | | | | |
| GPT-5 | 89.0 | 72.1 | 86.7 | 71.0 |
| GPT-5 (Non-Reasoning) | 86.3 | 49.5 | 52.4 | 40.7 |
| Gemini 2.5 Pro | 87.4 | 63.3 | 59.0 | 59.7 |
| Gemini 2.5 Flash | 83.4 | 57.3 | 57.1 | 49.8 |
| Claude Sonnet 4 | 85.5 | 49.0 | 36.2 | 46.8 |
| Claude 3.7 Sonnet Thinking | 85.8 | 53.3 | 34.3 | 45.5 |
| Llama 4 Maverick | 85.3 | 35.1 | 18.1 | 30.2 |
| Mistral Medium 3.1 | 84.5 | 34.8 | 27.6 | 27.7 |
| Text-only Models | | | | |
| GPT-OSS 120B | 84.2 | 65.5 | 56.1 | 57.8 |
| DeepSeek R1 | 86.9 | 57.4 | 38.6 | 47.1 |

(3) Variations in Competencies Across Models. Our skill framework exposes model-specific strengths and failures that a single aggregate score hides. Overall, 23 of the 26 models evaluated (Appendix B.3) score lower on either analysis or design QAs than on implementation 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).

(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).

(4) Variations in domain expertise. Our analysis reveals that LLMs develop specialized expertise across different domains. For instance, within “Implement”-Style questions, Llama 4 Maverick performs well on Computing Domains & Workloads and struggles on Modeling & Simulation, while Mistral Medium 3.1 exhibits the opposite behavior. Importantly, Llama 4 Maverick and Mistral Medium 3.1 overall performed similarly on “Implement”-Style questions, suggesting model capabilities are 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 all frontier models across all skills are shown in Appendix B.2.

(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., 2024a). Multimodal models exhibit an average 6% drop in accuracy from text-only free-response to image-only free-response questions (Appendix Table 6). 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 CASE STUDY: DESIGNING A MEMORY CONTROLLER

To illustrate how QUARCH-REASONING performance translates to real architectural decision-making, we conduct a case study 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.

Figure 7: 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). The prompt lists (Appendix D.11) all design parameters and explicitly asks the model to analyze the workload’s 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 inferences 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 7 reports the mean energy efficiency (power¹, normalized within each workload) of each model’s best controller designs across the 10 runs versus their QUARCH-REASONING performance.

We observe a strong, 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.

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; Vas et al., 2024; Pinckney et al., 2025b), the approach is still relatively new. We therefore validate the reliability 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 from the 10 models under evaluation in Sec. 4.1. We tasked a cohort of 11 domain experts

in computer architecture and hardware design to manually grade the resultant 908 responses 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. To control for LLM judge stochasticity, all judge evaluations in Sec. 4.1 and Appendix B.4 are likewise performed 3x and the majority vote taken.

LLM judges agree with human experts. We observe an agreement rate of 85.35% between LLM judges and humans (Fig. 8), when using Claude 3.7 Sonnet as the judge. We compare this agreement rate with the rate that expert humans disagree on verdicts. 84 of the 908 responses required a third expert to adjudicate between a correct and incorrect vote, corresponding to a human-to-human agreement rate of 90.7%. Since this agreement rate is broadly comparable to the frequency with which LLM-as-a-Judge consensus 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 LLMs as judges, and expert human grading on FRQ. majority consensus rates are included in Appendix B.6

Figure 8: Confusion matrix comparing LLM-as-a-Judge with domain-expert human grading on FRQ.

5 RELATED WORK

Software. Function-level code efficiency benchmarks (Du et al., 2024; Huang et al., 2024; Shypula et al., 2024; Waghjale et al., 2024) and domain-focused performance tasks (Press et al., 2025; 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., 2024b; 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., 2024; 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, testing whether models can reason about architectural principles and 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., 2016; Clark et al., 2019; Hendrycks et al., 2020; Rein et al., 2023; 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, 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.

¹Non-multimodal models cannot generate responses for the multimodal proportion of sampled questions.

540 ETHICS STATEMENT

541

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

554 REPRODUCIBILITY STATEMENT

555

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

560

561 REFERENCES

562

- 563 Elisavet Lydia Alvanaki, Kevin Lee, and Luca P Carloni. Sldb: An end-to-end heterogeneous
564 system-on-chip benchmark suite for llm-aided design. In 2025 IEEE International Conference
565 on LLM-Aided Design (ICLAD), pp. 227–234. IEEE, 2025.
- 566 Geoffrey Blake, Ronald G. Dreslinski, and Trevor Mudge. A survey of multicore processors. IEEE
567 Signal Processing Magazine, 26(6):26–37, 2009. doi: 10.1109/MSP.2009.934110.
- 568 Jason Blocklove, Siddharth Garg, Ramesh Karri, and Hammond Pearce. Chip-chat: Challenges and
569 opportunities in conversational hardware design. In 2023 ACM/IEEE 5th Workshop on Machine
570 Learning for CAD (MLCAD), pp. 1–6. IEEE, 2023.
- 571 Luca P. Carloni. From latency-insensitive design to communication-based system-level design. Pro-
572 ceedings of the IEEE, 103(11):2133–2151, 2015. doi: 10.1109/JPROC.2015.2480849.
- 573 Kaiyan Chang, Ying Wang, Haimeng Ren, Mengdi Wang, Shengwen Liang, Yinhe Han, Huawei Li,
574 and Xiaowei Li. Chipgpt: How far are we from natural language hardware design. arXiv preprint
575 arXiv:2305.14019, 2023.
- 576 Kaiyan Chang, Zhirong Chen, Yunhao Zhou, Wenlong Zhu, Kun Wang, Haobo Xu, Cangyuan Li,
577 Mengdi Wang, Shengwen Liang, Huawei Li, et al. Natural language is not enough: Bench-
578 marking multi-modal generative ai for verilog generation. In Proceedings of the 43rd IEEE/ACM
579 International Conference on Computer-Aided Design, pp. 1–9, 2024a.
- 580 Kaiyan Chang, Zhirong Chen, Yunhao Zhou, Wenlong Zhu, Haobo Xu, Cangyuan Li, Mengdi Wang,
581 Shengwen Liang, Huawei Li, Yinhe Han, et al. Natural language is not enough: Benchmarking
582 multi-modal generative ai for verilog generation. arXiv preprint arXiv:2407.08473, 2024b.
- 583 Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina
584 Toutanova. Boolq: Exploring the surprising difficulty of natural yes/no questions. arXiv preprint
585 arXiv:1905.10044, 2019.
- 586 Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser,
587 Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John
588 Schulman. Training Verifiers to Solve Math Word Problems, November 2021.
- 589 William J Dally, Yatish Turakhia, and Song Han. Domain-specific hardware accelerators. Commu-
590 nications of the ACM, 63(7):48–57, 2020.

593

- 594 Matthew DeLorenzo, Animesh Basak Chowdhury, Vasudev Gohil, Shailja Thakur, Ramesh Karri,
595 Siddharth Garg, and Jeyavijayan Rajendran. Make every move count: Llm-based high-quality rtl
596 code generation using mcts. arXiv preprint arXiv:2402.03289, 2024a.
597
- 598 Matthew DeLorenzo, Vasudev Gohil, and Jeyavijayan Rajendran. Creativeval: Evaluating creativity
599 of llm-based hardware code generation. arXiv preprint arXiv:2404.08806, 2024b.
600
- 601 Elizabeth Dinella, Satish Chandra, and Petros Maniatis. Crqbench: A benchmark of code reason-
602 ing questions. arXiv preprint arXiv:2408.08453, 2024. doi: 10.48550/arXiv.2408.08453. URL
603 <https://arxiv.org/abs/2408.08453>.
604
- 605 Mingzhe Du, Anh Tuan Luu, Bin Ji, Qian Liu, and See-Kiong Ng. Mercury: A code ef ciency
606 benchmark for code large language models. In NeurIPS Datasets and Benchmarks, 2024. URL
607 <https://arxiv.org/abs/2402.07844>.
- 608 Ravit Golan Duncan. The role of domain-speci c knowledge in generative reasoning about compli-
609 cated multileveled phenomena. *Cognition and Instruction*, 25(4):271–336, 2007.
610
- 611 Hadi Esmaeilzadeh, Emily Blem, Renee St. Amant, Karthikeyan Sankaralingam, and Doug Burger.
612 Dark silicon and the end of multicore scaling. In Proceedings of the 38th annual international
613 symposium on Computer architecture, pp. 365–376, 2011.
614
- 615 Weimin Fu, Kaichen Yang, Raj Gautam Dutta, Xiaolong Guo, and Gang Qu. Llm4sechw: Lever-
616 aging domain-speci c large language model for hardware debugging. In 2023 Asian Hardware
617 Oriented Security and Trust Symposium (AsianHOST), pp. 1–6. IEEE, 2023.
- 618 Alex Gu, Baptiste Roziere, Hugh James Leather, Armando Solar-Lezama, Gabriel Synnaeve, and
619 Sida Wang. Cruxeval: A benchmark for code reasoning, understanding and execution. In
620 Proceedings of the 41st International Conference on Machine Learning (ICML), volume 235
621 of Proceedings of Machine Learning Research, pp. 16568–16621. PMLR, July 2024. URL
622 <https://proceedings.mlr.press/v235/gu24c.html>.
623
- 624 Xinyi He, Qian Liu, Mingzhe Du, Lin Yan, Zhijie Fan, Yiming Huang, Zejian Yuan, and Zejun Ma.
625 Swe-perf: Can language models optimize code performance on real-world repositories? arXiv
626 preprint arXiv:2507.12415, 2025.
- 627 Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and
628 Jacob Steinhardt. Measuring massive multitask language understanding. arXiv preprint
629 arXiv:2009.03300, 2020.
630
- 631 Dong Huang, Yuhao Qing, Weiyi Shang, Heming Cui, and Jie M. Zhang. Ef bench: Benchmarking
632 the ef ciency of automatically generated code. In NeurIPS Datasets and Benchmarks, 2024. URL
633 <https://arxiv.org/abs/2402.02037>.
634
- 635 Patrick Huber, Armen Aghajanyan, Barlas Oguz, Dmytro Okhonko, Scott Yih, Sonal Gupta, and
636 Xilun Chen. CCQA: A New Web-Scale Question Answering Dataset for Model Pre-Training.
637 In Marine Carpuat, Marie-Catherine de Marneffe, and Ivan Vladimir Meza Ruiz (eds.), Find-
638 ings of the Association for Computational Linguistics: NAACL 2022, pp. 2402–2420, Seattle,
639 United States, July 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.
640 ndings-naacl.184.
- 641 Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, O r Press, and Karthik
642 Narasimhan. Swe-bench: Can language models resolve real-world github issues? arXiv preprint
643 arXiv:2310.06770, 2023.
644
- 645 Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, O r Press, and Karthik R
646 Narasimhan. SWE-bench: Can language models resolve real-world github issues? In The Twelfth
647 International Conference on Learning Representations, 2024. URL <https://openreview.net/forum?id=VTF8yNQM66>.

- 648 Qiao Jin, Bhuwan Dhingra, Zhengping Liu, William Cohen, and Xinghua Lu. PubMedQA: A
649 Dataset for Biomedical Research Question Answering. In Kentaro Inui, Jing Jiang, Vincent Ng,
650 and Xiaojun Wan (eds.), Proceedings of the 2019 Conference on Empirical Methods in Natural
651 Language Processing and the 9th International Joint Conference on Natural Language Process-
652 ing (EMNLP-IJCNLP), pp. 2567–2577, Hong Kong, China, November 2019. Association for
653 Computational Linguistics. doi: 10.18653/v1/D19-1259.
- 654 Matthias Jung, Christian Weis, and Norbert Wehn. Dramsys: A exible dram subsystem design
655 space exploration framework. IPSJ Transactions on System and LSI Design Methodology, 8:
656 63–74, 2015.
- 657 Stefan H Krieger. Domain knowledge and the teaching of creative legal problem solving. Clinical
658 L. Rev., 11:149, 2004.
- 660 Srivatsan Krishnan, Amir Yazdanbakhsh, Shvetank Prakash, Jason Jabbour, Ikechukwu Uchendu,
661 Susobhan Ghosh, Behzad Boroujerdian, Daniel Richins, Devashree Tripathy, Aleksandra Faust,
662 et al. Archgym: An open-source gymnasium for machine learning assisted architecture design.
663 In Proceedings of the 50th Annual International Symposium on Computer Architecture, pp. 1–16,
664 2023.
- 665 Seongyun Lee, Seungone Kim, Sue Park, Geewook Kim, and Minjoon Seo. Prometheus-vision:
666 Vision-language model as a judge for ne-grained evaluation. In Findings of the association for
667 computational linguistics ACL 2024, pp. 11286–11315, 2024.
- 669 Linyi Li, Shijie Geng, Zhenwen Li, Yibo He, Hao Yu, Ziyue Hua, Guanghan Ning,
670 Siwei Wang, Tao Xie, and Hongxia Yang. In bench: Evaluating the question-
671 answering capabilities of code large language models. In Advances in Neural Infor-
672 mation Processing Systems 37 (NeurIPS 2024), Datasets and Benchmarks Track, 2024.
673 URL [https://proceedings.neurips.cc/paper_files/paper/2024/
674 hash/e888eb9400fe14bb70e057aa1d719188-Abstract-Datasets_and_
675 Benchmarks_Track.html](https://proceedings.neurips.cc/paper_files/paper/2024/hash/e888eb9400fe14bb70e057aa1d719188-Abstract-Datasets_and_Benchmarks_Track.html).
- 676 Mingjie Liu, Teodor-Dumitru Ene, Robert Kirby, Chris Cheng, Nathaniel Pinckney, Rongjian Liang,
677 Jonah Alben, Himyanshu Anand, Sanmitra Banerjee, Ismet Bayraktaroglu, et al. Chipnemo:
678 Domain-adapted llms for chip design. arXiv preprint arXiv:2311.00176, 2023a.
- 679 Mingjie Liu, Nathaniel Pinckney, Brucek Khailany, and Haoxing Ren. Verilogeval: Evaluating large
680 language models for verilog code generation. In 2023 IEEE/ACM International Conference on
681 Computer Aided Design (ICCAD), pp. 1–8. IEEE, 2023b.
- 683 Shang Liu, Wenji Fang, Yao Lu, Qijun Zhang, Hongce Zhang, and Zhiyao Xie. Rtlcoder: Outper-
684 forming gpt-3.5 in design rtl generation with our open-source dataset and lightweight solution. In
685 2024 IEEE LLM Aided Design Workshop (LAD), pp. 1–5. IEEE, 2024.
- 686 LlamaIndex. Llamaparse: A genai-native document parsing platform. Software, 2025. URL
687 <https://cloud.llamaindex.ai/parse>. Accessed: 2025-09-18.
- 688 Bhabesh Mali, Karthik Maddala, Vatsal Gupta, Sweeya Reddy, Chandan Karfa, and Ramesh Karri.
689 Chiraag: Chatgpt informed rapid and automated assertion generation. In 2024 IEEE Computer
690 Society Annual Symposium on VLSI (ISVLSI), pp. 680–683. IEEE, 2024.
- 692 Oscar Marias, Benno Krojer, and Aishwarya Agrawal. Improving automatic vqa evaluation using
693 large language models. In Proceedings of the AAAI Conference on Arti cial Intelligence, vol-
694 ume 38, pp. 4171–4179, 2024.
- 695 Xingyu Meng, Amisha Srivastava, Ayush Arunachalam, Avik Ray, Pedro Henrique Silva, Rafail
696 Psiakis, Yiorgos Makris, and Kanad Basu. Nspg: Natural language processing-based security
697 property generator for hardware security assurance. In Proceedings of the 61st ACM/IEEE Design
698 Automation Conference, pp. 1–6, 2024.
- 700 Ivo Petrov Nikola Jovanoi Martin Vechev Mislav Balunoi Jasper Dekoninck. Matharena: Eval-
701 uating llms on uncontaminated math competitions. <https://matharena.ai/>, 2025. SRI
Lab, ETH Zurich.

- 702 Madhav Nair, Rajat Sadhukhan, and Debdeep Mukhopadhyay. Generating secure hardware using
703 chatgpt resistant to cwes. Cryptology ePrint Archive, 2023.
704
- 705 Dung Manh Nguyen, Thang Chau Phan, Nam Le Hai, Tien-Thong Doan, Nam V. Nguyen, Quang
706 Pham, and Nghi D. Q. Bui. Codemmlu: A multi-task benchmark for assessing code understanding
707 & reasoning capabilities of codellms. In International Conference on Learning Representations
708 (ICLR), 2025. URL <https://openreview.net/forum?id=CahIEKC5Q>.
- 709 OpenAI. Introducing swe-bench veri ed. Blog post, OpenAI, August 2024. URL <https://openai.com/index/introducing-swe-bench-verified/>. Updated February 24,
710 2025.
711
- 712 Krista Opsahl-Ong, Michael J Ryan, Josh Purtell, David Broman, Christopher Potts, Matei Zaharia,
713 and Omar Khattab. Optimizing instructions and demonstrations for multi-stage language model
714 programs. arXiv preprint arXiv:2406.11695, 2024.
715
- 716 Anne Ouyang, Simon Guo, Simran Arora, Alex L Zhang, William Hu, Christopher R. and Azalia
717 Mirhoseini. Kernelbench: Can llms write ef cient gpu kernels? arXiv preprint arXiv:2502.10517,
718 2025.
719
- 720 Hammond Pearce, Benjamin Tan, Baleegh Ahmad, Ramesh Karri, and Brendan Dolan-Gavitt. Ex-
721 amining zero-shot vulnerability repair with large language models. In 2023 IEEE Symposium on
722 Security and Privacy (SP), pp. 2339–2356. IEEE, 2023.
723
- 724 Zehua Pei, Hui-Ling Zhen, Mingxuan Yuan, Yu Huang, and Bei Yu. Betterv: Controlled verilog
725 generation with discriminative guidance. arXiv preprint arXiv:2402.03375, 2024.
- 726 Long Phan, Alice Gatti, Ziwen Han, Nathaniel Li, Josephina Hu, Hugh Zhang, Chen Bo Calvin
727 Zhang, Mohamed Shaaban, John Ling, Sean Shi, et al. Humanity's last exam. arXiv preprint
728 arXiv:2501.14249, 2025.
729
- 730 Nathaniel Pinckney, Christopher Batten, Mingjie Liu, Haoxing Ren, and Brucek Khailany. Revisit-
731 ing verilogeval: A year of improvements in large-language models for hardware code generation.
732 ACM Transactions on Design Automation of Electronic Systems, 2025a.
- 733 Nathaniel Pinckney, Chenhui Deng, Chia-Tung Ho, Yun-Da Tsai, Mingjie Liu, Wenfei Zhou, Brucek
734 Khailany, and Haoxing Ren. Comprehensive verilog design problems: A next-generation bench-
735 mark dataset for evaluating large language models and agents on rtl design and veri cation. arXiv
736 preprint arXiv:2506.14074, 2025b.
737
- 738 Ori Press, Brandon Amos, Haoyu Zhao, Yikai Wu, Samuel K. Ainsworth, Dominik Krupke, Patrick
739 Kidger, Touqir Sajed, Bartolomeo Stellato, Jisun Park, Nathanael Bosch, Eli Meril, Albert Steppi,
740 Arman Zharmagambetov, Fangzhao Zhang, David Perez-Pineiro, Alberto Mercurio, Ni Zhan,
741 Talor Abramovich, Kilian Lieret, Hanlin Zhang, Shirley Huang, Matthias Bethge, and O r Press.
742 Algotune: Can language models speed up general-purpose numerical programs?, 2025. URL
743 <https://arxiv.org/abs/2507.15887>.
- 744 Jiarui Qin, Yunjia Xi, Junjie Huang, Renting Rui, Di Yin, Weiwen Liu, Yong Yu, Weinan Zhang,
745 and Xing Sun. APTBench: Benchmarking Agentic Potential of Base LLMs During Pre-Training,
746 October 2025.
747
- 748 Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. SQuAD: 100,000+ Ques-
749 tions for Machine Comprehension of Text. In Jian Su, Kevin Duh, and Xavier Carreras (eds.),
750 Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, pp.
751 2383–2392, Austin, Texas, November 2016. Association for Computational Linguistics. doi:
752 10.18653/v1/D16-1264.
- 753 Vijay Janapa Reddi and Amir Yazdanbakhsh. Architecture 2.0: Foundations of Arti cial In-
754 telligence Agents for Modern Computer System Design . Computer, 58(02):116–124, Febru-
755 ary 2025. ISSN 1558-0814. doi: 10.1109/MC.2024.3521641. URL [https://doi.
ieeecomputersociety.org/10.1109/MC.2024.3521641](https://doi.ieeecomputersociety.org/10.1109/MC.2024.3521641).

- 756 David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Di-
757 rani, Julian Michael, and Samuel R Bowman. Gpqa: A graduate-level google-proof q&a bench-
758 mark. arXiv preprint arXiv:2311.12022, 2023.
- 759 Anna Rogers, Matt Gardner, and Isabelle Augenstein. Qa dataset explosion: A taxonomy of nlp
760 resources for question answering and reading comprehension. ACM Computing Surveys, 55(10):
761 1–45, 2023.
- 762 Alberto Sangiovanni-Vincentelli. Quo vadis, sld? reasoning about the trends and challenges of
763 system level design. Proceedings of the IEEE, 95(3):467–506, 2007. doi: 10.1109/JPROC.2006.
764 890107.
- 765 Alexander Shypula, Aman Madaan, Yimeng Zeng, Uri Alon, Jacob Gardner, Milad Hashemi, Gra-
766 ham Neubig, Parthasarathy Ranganathan, Osbert Bastani, and Amir Yazdanbakhsh. Learning
767 performance-improving code edits. In ICLR, 2024. URL [https://arxiv.org/pdf/2302.
768 07867](https://arxiv.org/pdf/2302.07867).
- 769 Shailja Thakur, Baleegh Ahmad, Hammond Pearce, Benjamin Tan, Brendan Dolan-Gavitt, Ramesh
770 Karri, and Siddharth Garg. Verigen: A large language model for verilog code generation. ACM
771 Transactions on Design Automation of Electronic Systems, 29(3):1–31, 2024.
- 772 Adam Trischler, Tong Wang, Xingdi Yuan, Justin Harris, Alessandro Sordoni, Philip Bachman, and
773 Kaheer Suleman. Newsqa: A machine comprehension dataset. arXiv preprint arXiv:1611.09830,
774 2016.
- 775 YunDa Tsai, Mingjie Liu, and Haoxing Ren. Rtl xer: Automatically fixing rtl syntax errors with
776 large language model. In Proceedings of the 61st ACM/IEEE Design Automation Conference, pp.
777 1–6, 2024.
- 778 Arya Tschand, Muhammad Awad, Ryan Swann, Kesavan Ramakrishnan, Jeffrey Ma, Keith Low-
779 erty, Ganesh Dasika, and Vijay Janapa Reddi. Swizzleperf: Hardware-aware llms for gpu kernel
780 performance optimization. arXiv preprint arXiv:2508.20258, 2025.
- 781 Vals AI, Inc. Vals.ai benchmarks. <https://www.vals.ai/benchmarks>, 2025. Accessed:
782 2025-09-24.
- 783 Siddhant Waghjale, Vishruth Veerendranath, Zora Zhiruo Wang, and Daniel Fried. ECCO: Can we
784 improve model-generated code efficiency without sacrificing functional correctness? arXiv, 2024.
785 URL <https://arxiv.org/abs/2407.14044>.
- 786 Gwok-Waa Wan, Shengchu Su, Ruihu Wang, Qixiang Chen, Sam-Zaak Wong, Mengnv Xing, Hefei
787 Feng, Yubo Wang, Yinan Zhu, Jingyi Zhang, et al. Fixme: Towards end-to-end benchmarking of
788 llm-aided design verification. arXiv preprint arXiv:2507.04276, 2025.
- 789 Xingyao Wang, Boxuan Li, Yufan Song, Frank F. Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan,
790 Yueqi Song, Bowen Li, Jaskirat Singh, Hoang H. Tran, Fuqiang Li, Ren Ma, Mingzhang Zheng,
791 Bill Qian, Yanjun Shao, Niklas Muennighoff, Yizhe Zhang, Binyuan Hui, Junyang Lin, Robert
792 Brennan, Hao Peng, Heng Ji, and Graham Neubig. Openhands: An open platform for AI soft-
793 ware developers as generalist agents. In The Thirteenth International Conference on Learning
794 Representations, 2025. URL <https://openreview.net/forum?id=OJd3ayDDoF>.
- 795 Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weiming
796 Ren, Aaran Arulraj, Xuan He, Ziyang Jiang, et al. Mmlu-pro: A more robust and challenging
797 multi-task language understanding benchmark. arXiv preprint arXiv:2406.01574, 2024a.
- 798 Zhihai Wang, Zijie Geng, Zhaojie Tu, Jie Wang, Yuxi Qian, Zhexuan Xu, Ziyang Liu, Siyuan Xu,
799 Zhentao Tang, Shixiong Kai, et al. Benchmarking end-to-end performance of ai-based chip place-
800 ment algorithms. arXiv preprint arXiv:2407.15026, 2024b.
- 801 Bing-Yue Wu, Utsav Sharma, Sai Rahul Dhanvi Kankipati, Ajay Yadav, Bintu Kappil George,
802 Sai Ritish Guntupalli, Austin Rovinski, and Vidya A Chhabria. Eda corpus: A large language
803 model dataset for enhanced interaction with openroad. arXiv preprint arXiv:2405.06676, 2024a.
- 804
- 805
- 806
- 807
- 808
- 809

810 Haoyuan Wu, Zhuolun He, Xinyun Zhang, Xufeng Yao, Su Zheng, Haisheng Zheng, and Bei Yu.
811 Chateda: A large language model powered autonomous agent for eda. *IEEE Transactions on*
812 *Computer-Aided Design of Integrated Circuits and Systems*, 2024b.

813
814 Wm A Wulf and Sally A McKee. Hitting the memory wall: Implications of the obvious. *ACM*
815 *SIGARCH computer architecture news*, 23(1):20–24, 1995.

816 John Yang, Carlos E Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik R Narasimhan,
817 and O r Press. SWE-agent: Agent-computer interfaces enable automated software engineering.
818 In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL
819 <https://arxiv.org/abs/2405.15793>.

820
821 Xufeng Yao, Haoyang Li, Tsz Ho Chan, Wenyi Xiao, Mingxuan Yuan, Yu Huang, Lei Chen, and
822 Bei Yu. Hdldebugger: Streamlining hdl debugging with large language models. *arXiv preprint*
823 *arXiv:2403.11671*, 2024.

824 Yongan Zhang, Zhongzhi Yu, Yonggan Fu, Cheng Wan, and Yingyan Celine Lin. Mg-verilog:
825 Multi-grained dataset towards enhanced llm-assisted verilog generation. In *2024 IEEE LLM Aided*
826 *Design Workshop (LAD)*, pp. 1–5. IEEE, 2024.

827
828 Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang,
829 Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. Judging llm-as-a-judge with mt-bench and
830 chatbot arena. *Advances in Neural Information Processing Systems*, 36:46595–46623, 2023.

831
832 Haoxi Zhong, Chaojun Xiao, Cunchao Tu, Tianyang Zhang, Zhiyuan Liu, and Maosong Sun. JEC-
833 QA: A Legal-Domain Question Answering Dataset. *Proceedings of the AAAI Conference on Arti-*
834 *cial Intelligence*, 34(05):9701–9708, April 2020. ISSN 2374-3468. doi: 10.1609/aaai.v34i05.
6519.

835
836 Chunting Zhou, Pengfei Liu, Puxin Xu, Srinivasan Iyer, Jiao Sun, Yuning Mao, Xuezhe Ma, Avia
837 Efrat, Ping Yu, Lili Yu, et al. Lima: Less is more for alignment. *Advances in Neural Information*
838 *Processing Systems*, 36:55006–55021, 2023.

839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863

864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917

APPENDIX

Table of Contents

| | | |
|-------|---|----|
| A | LLM Usage | 19 |
| B | Additional Results | 19 |
| B.1 | QuArch Crowdsourcing Website | 19 |
| B.2 | Topic-wise Further Characterization | 23 |
| B.3 | Full Model Results by Skill | 23 |
| B.4 | Model Performance by Modality | 23 |
| B.5 | Partially Correct Judgments | 25 |
| B.6 | Comparing Human Domain Experts to LLM-as-a-Judge | 26 |
| C | Example Questions | 28 |
| C.1 | Examples of Question Skills Taxonomy | 28 |
| C.1.1 | Recall Question | 28 |
| C.1.2 | Analysis Question | 28 |
| C.1.3 | Design Question | 29 |
| C.1.4 | Implementation Question | 30 |
| C.2 | Failure Mode 1: Struggles with Architecture–Semantics of Code Execution | 31 |
| C.2.1 | Example 1 | 31 |
| C.2.2 | Example 2 | 33 |
| C.2.3 | Example 3 | 35 |
| C.2.4 | Example 4 | 38 |
| C.3 | Failure Mode 2: Assuming Unconventional Architectural Properties | 39 |
| C.3.1 | Example 1 | 39 |
| C.3.2 | Example 2 | 41 |
| C.3.3 | Example 3 | 43 |
| C.4 | Failure Mode 3: Modeling and Tracking System State | 45 |
| C.4.1 | Example 1 | 45 |
| C.4.2 | Example 2 | 48 |
| C.5 | Failure Mode 4: Sensitivity to QA Modality | 50 |
| C.5.1 | Example 1 | 50 |
| C.5.2 | Example 2 | 52 |
| C.6 | Successful Model Response | 53 |
| C.7 | Example Questions for Case Study | 54 |
| D | Prompt Templates | 57 |
| D.1 | LLM Prompt for Student LLM | 57 |
| D.2 | LLM Prompt for MCQ Response | 58 |
| D.3 | LLM Prompt for FRQ LLM-as-a-Judge | 58 |
| D.4 | LLM Prompt for Skills Classification | 59 |
| D.6 | LLM Prompt for MCQ Generation | 61 |
| D.7 | LLM Prompts for MCQ Filtering | 62 |
| D.8 | LLM Prompt for Exam Question Text Extraction | 63 |
| D.9 | LLM Prompt for Exam Question Image Extraction | 65 |

| | | | |
|-----|------|--|----|
| 918 | D.10 | LLM Prompt for Exam Question Verification | 66 |
| 919 | D.11 | LLM Prompt for Memory Controller Design Case Study | 66 |
| 920 | | | |
| 921 | | | |
| 922 | | | |
| 923 | | | |
| 924 | | | |
| 925 | | | |
| 926 | | | |
| 927 | | | |
| 928 | | | |
| 929 | | | |
| 930 | | | |
| 931 | | | |
| 932 | | | |
| 933 | | | |
| 934 | | | |
| 935 | | | |
| 936 | | | |
| 937 | | | |
| 938 | | | |
| 939 | | | |
| 940 | | | |
| 941 | | | |
| 942 | | | |
| 943 | | | |
| 944 | | | |
| 945 | | | |
| 946 | | | |
| 947 | | | |
| 948 | | | |
| 949 | | | |
| 950 | | | |
| 951 | | | |
| 952 | | | |
| 953 | | | |
| 954 | | | |
| 955 | | | |
| 956 | | | |
| 957 | | | |
| 958 | | | |
| 959 | | | |
| 960 | | | |
| 961 | | | |
| 962 | | | |
| 963 | | | |
| 964 | | | |
| 965 | | | |
| 966 | | | |
| 967 | | | |
| 968 | | | |
| 969 | | | |
| 970 | | | |
| 971 | | | |

972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999

Figure 9: QuArch Website Home Page

1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025

A 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 RESULTS

B.1 QUARCH CROWDSOURCING WEBSITE

To provide a centralized location to crowdsource questions and exams, we created a QuArch website, shown in Figure 9. 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 10. Users submit the question, four answer options, the correct answer option, and a rationale for the correct answer, as shown in Figure 11. While the user-submitted question is formatted by default in multiple-choice style, we instruct them to not submit questions where the correct answer is an “all of the above” or “none of the above.” This enables us to concatenate the correct answer with the rationale and format the question as a free-response as well.

Once a question is written, we enable users to seamlessly test four non-frontier LLMs on correctness, shown in Figure 12. The question and potential answer choices is sent to the respective model within the MCQ Prompt shown in Appendix D.2. Users are presented with whether the model gets the question correct and its response. We explicitly do not want to give the users access to test their question on frontier LLMs to prevent them from overfitting to only specific LLMs. By giving users signal on whether non-frontier LLMs fail, they can create difficult and correct questions that

1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079

Figure 10: QuArch Website Submit Question Instructions

Figure 11: QuArch Website Submit Question Portal

1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133

Figure 12: QuArch Website LLM Testing on Question Submission

Figure 13: QuArch Website Question Submission Scoring

1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187

Figure 14: QuArch Website Admin Approvals

Figure 15: QuArch Website Exam Collection Portal

challenge a broad range of LLMs. The LLM performance is used to assign a score for the question, where making more frontier models fail corresponds to more points, shown in Figure 13. The website tracks submissions for each user, and we compile a leaderboard for users with the most cumulative points and question submissions.

Once a question is submitted, we created an admin approval portal, shown in Figure 14. The approval process involves checking the questions for missing assumptions, poor relevance, or an insufficient rationale. Admins also reproduce each answer manually to ensure that the question-answer pair is objective and correct. Once a submission question is approved, it is added to the QUARCH benchmark set and the user that submitted the question is notified.

Lastly, users can also submit exams in the website, shown in Figure 15. We will then parse out the questions, context images, and answers from these computer architecture course exams using the offline methodology described in Sec. 2.2. Cumulatively, the questions collected from the QuArch crowdsourcing website are taxonomized by skill and funneled into the QUARCH-Reasoning benchmark.

B.2 TOPIC-WISE FURTHER CHARACTERIZATION

Figure 16 visualizes the topic-wise performance of the frontier models on QUARCH. As performance across all models on QUARCH-Recall is very high, very little topic-wise variability can be seen. However, in the higher-level skills (Analyze, Design, and Implement), models exhibit surprising heterogeneity in per-topic performance.

B.3 FULL MODEL RESULTS BY SKILL

Table 4 provides the complete set of results for all 26 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. While we do not discuss individual results here, the table captures the broader landscape of model performance and makes clear the varying degrees of capability across skills that are critical for computer architecture reasoning.

Performance of Small Language Models

Small language models (SLMs) keep pace with LLMs on recall-style questions, but their performance drops sharply on analyze, design, and implement tasks—especially when multimodal reasoning is required. This gap suggests that parameter scale (and associated capacity for long-horizon reasoning and state tracking) matters far more for higher-order architectural problem solving than for fact 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 recall to SLMs, escalate complex reasoning to LLMs, and bolster SLMs with retrieval and simulators rather than relying on scale alone.

We evaluate the performance of SLMs (defined as ≤ 30 B parameters) vs LLMs (defined as ≥ 70 B parameters or proprietary frontier models) on the taxonomy of QUARCH questions. SLMs exhibit a 13% drop in Recall accuracy, but a 27% drop on Analyze, 26% drop on Design, and 25% drop on Implement questions compared to LLMs. This indicates that while SLMs can replicate factual knowledge nearly as well as their larger counterparts, they falter on the higher-order reasoning and synthesis that are core to the field of computer architecture. These results imply that scaling up—or supplementing smaller models with external reasoning tools—is critical for bridging the gap in advanced architectural skills.

B.4 MODEL PERFORMANCE BY MODALITY

In addition to the main set of frontier models, we conduct a comprehensive evaluation on 26 models spanning both small language models (SLMs) and large language models (LLMs), and breakdown performance by question modality. Results are summarized in Table B.4. This broader analysis highlights consistent trends across scale, including significant gaps in higher-order reasoning and

1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295

Figure 16: 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 4: Accuracy (%) of all evaluated models across the four skills in QUARCH. Best performing models in each category highlighted **rst**, **second**, and **third**.

| Model | QUARCH-REASONING | | | | Overall |
|--------------------------------------|------------------|---------|--------|-----------|---------|
| | Recall | Analyze | Design | Implement | |
| Closed-Source Multimodal Models | | | | | |
| GPT-5 | 89.0 | 72.1 | 86.7 | 71.0 | 72.4 |
| GPT-5 (Non-Reasoning) | 86.3 | 49.5 | 52.4 | 40.7 | 49.0 |
| GPT-4o | 84.1 | 28.4 | 12.4 | 22.9 | 27.5 |
| Gemini 2.5 Pro | 87.4 | 63.3 | 59.0 | 59.7 | 62.9 |
| Gemini 2.5 Flash | 83.4 | 57.3 | 57.1 | 49.8 | 56.8 |
| Claude Sonnet 4 | 85.5 | 49.0 | 36.2 | 46.8 | 48.4 |
| Claude 3.7 Sonnet Thinking | 85.8 | 53.3 | 34.3 | 45.5 | 52.1 |
| Mistral Medium 3.1 | 84.5 | 34.8 | 27.6 | 27.7 | 34.1 |
| Open-Source Multimodal Models | | | | | |
| Gemma 3 27B IT | 75.6 | 22.4 | 15.2 | 16.0 | 21.7 |
| Gemma 3 4B IT | 62.3 | 7.6 | 2.9 | 3.5 | 7.2 |
| Llama 4 Maverick | 85.3 | 35.1 | 18.1 | 30.2 | 34.2 |
| Llama 3.2 11B | 69.4 | 8.8 | 1.0 | 4.0 | 8.2 |
| Mistral Small 3.2 24B Instruct | 78.0 | 24.3 | 16.2 | 17.7 | 23.6 |
| Text-Only Models | | | | | |
| GPT-OSS 120B | 84.2 | 65.5 | 56.1 | 57.8 | 64.7 |
| DeepSeek R1 | 86.9 | 57.4 | 38.6 | 47.1 | 56.1 |
| Llama 3.3 70B | 80.4 | 25.8 | 0.0 | 13.7 | 24.2 |
| Llama 3.2 1B | 36.5 | 1.7 | 0.0 | 1.0 | 1.6 |
| Mistral Codestral 2508 | 75.3 | 29.7 | 5.3 | 16.7 | 28.1 |
| Mistral Devstral Medium | 81.9 | 29.0 | 3.5 | 22.5 | 27.7 |
| Kimi K2 0905 | 84.2 | 43.9 | 35.1 | 37.3 | 43.2 |
| Qwen3 Coder 480B A35B Instruct | 82.9 | 41.7 | 17.5 | 31.4 | 40.2 |
| Qwen3 235B A22B Thinking | 85.6 | 62.2 | 50.9 | 52.9 | 61.3 |
| Qwen3 235B A22B NonThinking Instruct | 86.5 | 56.4 | 42.1 | 47.1 | 55.3 |
| Qwen3 Next 80B A3B Thinking | 84.5 | 54.9 | 36.8 | 42.2 | 53.5 |
| Qwen3 30B A3B Thinking | 82.5 | 50.0 | 31.6 | 37.3 | 48.6 |
| Qwen3 Coder 30B A3B Instruct | 78.3 | 28.2 | 8.8 | 12.7 | 26.6 |

multimodal tasks compared to 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.

Table 6 provides detailed comparisons between text-only question performance, image-only performance, and image-text performance. See Sec 4 for interpretation and analysis of sensitivity to input modalities.

B.5 PARTIALLY CORRECT JUDGMENTS

Table 7 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. For example, 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/in-correct 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 would imply. 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 use-

Table 5: Per-generation accuracy (%) by evaluation type and modality. Best performing models in each category highlighted **rst**, **second**, and **third**.

| Model | Text-only | | Image-only | Image & Text |
|--|-----------|------|------------|--------------|
| | MCQ | FRQ | FRQ | FRQ |
| Closed-Source Multimodal Models | | | | |
| GPT-5 (High Effort) | 89.0 | 74.7 | 70.5 | 72.7 |
| GPT-5 | 86.4 | 53.8 | 44.8 | 49.4 |
| GPT-4o | 84.3 | 31.3 | 24.6 | 28.1 |
| 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 |
| 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 | - | - |

ful 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.6 COMPARING HUMAN DOMAIN EXPERTS TO LLM-AS-A-JUDGE

We rigorously validate the reliability of LLM judges used on QUARCH QAs compared to human expert evaluators in Sec. 4.5 and this section. We instruct LLM-as-a-Judge to reason about the accuracy of each freeform response with respect to the ground truth answer as though the response is from a student completing an academic exam (see prompts in Appendix D.3). The judge is instructed to grade each response as CORRECT, PARTIALLY-CORRECT, or INCORRECT. For our reported evaluations (Sec. 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 two purposes: it dis-incentivizes the judge from rounding up a nearly-correct answer to correct, and it enables analysis of fine-grained knowledge (Appendix B.5).

We generate multiple samples per question and multiple judgments per sample to control for model stochasticity, and report 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

1404 Table 6: Per-generation accuracy (%) on the QUARCH benchmark broken down by evaluation type
 1405 and modality. Best performing models in each category highlighted **rst**, **second**, and **third**.
 1406

| 1407 Model | 1408 Text-only | | 1408 Image-only | 1408 Image & Text |
|---------------------------------|----------------|----------|-----------------|-------------------|
| | 1408 MCQ | 1408 FRQ | 1408 FRQ | 1408 FRQ |
| 1409 Multimodal Models | | | | |
| 1410 GPT-5 (High Effort) | 89.0 | 74.7 | 70.5 | 72.7 |
| 1411 GPT-5 | 86.4 | 53.8 | 44.8 | 49.4 |
| 1412 Gemini 2.5 Pro | 87.5 | 63.9 | 62.1 | 63.0 |
| 1413 Gemini 2.5 Flash | 83.2 | 59.3 | 55.4 | 57.4 |
| 1414 Claude Sonnet 4 | 85.6 | 52.1 | 45.4 | 48.8 |
| 1415 Claude 3.7 Sonnet Thinking | 85.9 | 53.8 | 51.3 | 52.6 |
| 1416 Llama 4 Maverick | 85.9 | 36.1 | 32.1 | 34.2 |
| 1417 Mistral Medium 3.1 | 84.7 | 41.2 | 27.7 | 34.6 |
| 1418 Text-Only Models | | | | |
| 1419 GPT-OSS 120B | 84.1 | 65.3 | - | - |
| 1420 DeepSeek R1 | 87.1 | 56.2 | - | - |

1421

1422 Table 7: Addition of “Partially Correct” Judgments in LLM-as-a-judge rubric. Results are pass@1
 1423 and using a single LLM-as-a-judge response rather than from consensus.
 1424

| 1425 Model | 1425 Correct (%) | 1425 Partially Correct (%) | 1425 Incorrect (%) |
|--|------------------|----------------------------|--------------------|
| 1426 Closed-Source Multimodal Models | | | |
| 1427 GPT-5 | 70.4 | 19.2 | 10.2 |
| 1428 GPT-5 (Non-Reasoning) | 48.5 | 31.6 | 20.0 |
| 1429 GPT-4o | 28.5 | 40.2 | 31.3 |
| 1430 Gemini 2.5 Pro | 61.9 | 24.1 | 13.8 |
| 1431 Gemini 2.5 Flash | 56.7 | 26.7 | 16.5 |
| 1432 Claude Sonnet 4 | 49.6 | 31.8 | 18.6 |
| 1433 Claude 3.7 Sonnet Thinking | 52.1 | 30.7 | 16.9 |
| 1434 Mistral Medium 3.1 | 33.5 | 34.5 | 31.8 |
| 1435 Open-Source Multimodal Models | | | |
| 1436 Gemma 3 27B Instruct | 24.0 | 35.2 | 40.9 |
| 1437 Gemma 3 4B Instruct | 6.6 | 30.9 | 62.5 |
| 1438 Llama 4 Maverick | 33.7 | 37.9 | 28.5 |
| 1439 Llama 3.2 11B | 8.6 | 27.9 | 63.5 |
| 1440 Mistral Small 3.2 24B Instruct | 24.5 | 36.4 | 39.0 |
| 1441 Text-Only Models | | | |
| 1442 GPT OSS 120B | 66.7 | 17.5 | 15.8 |
| 1443 DeepSeek R1 | 55.9 | 25.2 | 18.8 |
| 1444 Llama 3.3 70B | 24.0 | 39.5 | 36.2 |
| 1445 Llama 3.2 1B | 0.3 | 13.4 | 86.3 |
| 1446 Mistral Codestral 2508 | 27.9 | 37.7 | 33.9 |
| 1447 Mistral Devstral Medium | 27.0 | 37.6 | 35.2 |
| 1448 Kimi K2 0905 | 45.4 | 25.8 | 28.8 |
| 1449 Qwen 3 Coder 480B Instruct | 40.7 | 34.8 | 24.5 |
| 1450 Qwen 3 235B A22B Thinking | 57.7 | 16.6 | 25.6 |
| 1451 Qwen 3 235B A22B NonThinking Instruct | 54.4 | 26.5 | 18.5 |
| 1452 Qwen 3 Next 80B Thinking | 56.1 | 20.6 | 23.2 |
| 1453 Qwen 3 30B A3B Thinking | 48.0 | 21.7 | 30.1 |
| 1454 Qwen 3 Coder 30B A3B Instruct | 28.9 | 34.7 | 36.2 |

1452

1453

1454 problem, two model samples are each judged by majority vote to be CORRECT, and the third sample
 1455 is majority vote INCORRECT, pass@k=1 on that problem is
 1456

1457 We use Claude-3.7-Thinking as our LLM judge for all benchmark evaluations in this paper. We
 compare Claude against Gemini-2.5-Pro as an alternate candidate for our judge LLM (using a con-

1458 sensus size of 3 for both), and observe Gemini-2.5-Pro achieves a slightly lower agreement rate of
 1459 84.73% against human experts, hence our adoption of Claude as our chosen judge. We also inspect
 1460 the frequency of necessitated tie-breaking under our consensus size of 3 (where a tie consists of
 1461 one correct and one incorrect verdict) across QUARCH's entire QA dataset and all models assessed
 1462 in the main text and appendix, and observe across 65,659 responses, the first two judgments from
 1463 Claude-3.7-Thinking matched 89.0% of the time (and hence did not require a third judgment to
 1464 adjudicate).

1465 In order to ascertain if LLM judge disagreements with human experts increased in frequency in
 1466 relation to grading difficulty, we additionally asked each human expert to assign each problem a
 1467 score between 1-5 on the difficulty of grading the question (not the difficulty of the question). This
 1468 trend did not emerge in our data; rather, the domain expertise and familiarity with academic content
 1469 in the human cohort led to 1 and 2 being the most frequently assigned scores for grading difficulty.
 1470 We believe this preliminary result leads to three potential directions for future work in alignment
 1471 between the performance of LLM-as-a-Judge and domain experts: (1) judge prompt optimization
 1472 by both domain experts and automated methods (Opsahl-Ong et al., 2024), such as by informing
 1473 the judge that students may try to earn extra points on a question they can't answer by including
 1474 relevant-sounding jargon to mimic understanding (we see this behavior exhibited by some SLMs),
 1475 (2) characterizing question difficulty in QUARCH and exploring whether harder questions are also
 1476 harder to accurately grade by both LLMs and humans, and (3) investigating the tradeoff across LLM
 1477 judge generation parameters between verdict determinism and verdict accuracy under consensus
 1478 when comparing against human expert verdicts as ground truth.

1479 C EXAMPLE QUESTIONS

1480 C.1 EXAMPLES OF QUESTION SKILLS TAXONOMY

1481 C.1.1 RECALL QUESTION

1482 Example 2: Storage Systems

1483 Question: Moving compute closer to the_ in solid state drives (SSDs) offers higher
 1484 bandwidth but introduces challenges in managing frequent errors.

1485 Answers:

- 1486 (a) controller
- 1487 (b) NAND dies
- 1488 (c) cache
- 1489 (d) DRAM

1490 Rationale: This question falls under the recall category as it requires retrieval of factual knowl-
 1491 edge about SSD architecture, specifically the trade-offs between bandwidth optimization and error
 1492 management when positioning compute resources relative to different storage components.
 1493

1494 C.1.2 ANALYSIS QUESTION

1500 Example 4: Systolic Array Analysis

1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511

1512
 1513
 1514
 1515
 1516
 1517
 1518
 1519
 1520
 1521
 1522
 1523
 1524
 1525
 1526
 1527
 1528
 1529
 1530
 1531
 1532
 1533
 1534
 1535
 1536
 1537
 1538
 1539
 1540
 1541
 1542
 1543
 1544
 1545
 1546
 1547
 1548
 1549
 1550
 1551
 1552
 1553
 1554
 1555
 1556
 1557
 1558
 1559
 1560
 1561
 1562
 1563
 1564
 1565

Context: Given arrays $X = [x_1; x_2; x_3; \dots; x_n]$ and $W = [w_1; w_2; w_3; \dots; w_k]$, you want to compute

$Y = [y_1; y_2; y_3; \dots; y_{(n+1-k)}]$ using the formula:

$$y_i = w_1x_{i-1} + w_2x_i + w_3x_{i+1} + \dots + w_kx_{i+k-1}$$

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 $y_i = w_1x_{i-1} + w_2x_i + w_3x_{i+1} + \dots + w_kx_{i+k-1}$. The number of PEs is equal to the number of Y values. Each PE computes one Y value. A single PE performs the computation $Y = W \cdot X$.

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 $y_2 = w_1x_2 + w_2x_3 + \dots + w_kx_{k+1}$. 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 DESIGN QUESTION

Example 4: Cache Partitioning and Associativity Trade-offs in Multicore Systems

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 IMPLEMENTATION QUESTION

Example 4: Linked-List Manipulation via Self-Modifying Code on EDSACjr

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 (`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.

Figure 18: Caption for image 2

Figure 17: Caption for image 1

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.

1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654

Question:

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

Correct Solution: 2:3

```
.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

Example: 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]
```

1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727

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 "in nite" 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 = in nite / 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 $M \times K$
- Second matrix B is $K \times N$
- Thus, resulting matrix Z is $M \times N$

```
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 = $N \times K$

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 ($M \times K$).

- 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 $N \times K$.
- 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 ($K \times N$).

- $B[k, n]$ is accessed within the innermost loop (for k in $[0, K)$).

1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748

- 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 ($M \times N$).

- $Z[m, n]$ 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.

1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768

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, directly determining cache hit rates through the stack distance model.

For matrix multiplication—the dominant kernel in scientific computing and deep learning—accurate reuse analysis guides critical optimization decisions: 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 $RI = N - K$ (reused only after all $N - K$ 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 $[0; N)$... 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 $N - K$ intervening references.

1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781

C.2.2 EXAMPLE 2

Example: 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);
        }
    }
}
```

1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835

```
    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 `length2`, 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,

1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846

- 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.

1847
1848
1849
1850
1851

Question Summary and Relevance: This question tests understanding of timing side-channel vulnerabilities in security-critical code, a fundamental 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.

1852
1853
1854
1855
1856
1857
1858
1859
1860

- **Secure computation:** Modern processors need to support constant-time operations for cryptographic code to prevent secret extraction through timing analysis.
- **Hardware–software security boundary:** Tests whether one understands that hardware guarantees (CMOV being constant-time) can be undermined by language-level ss.
- **Compiler behavior:** The short-circuit evaluation of && is a compiler optimization that can inadvertently break security guarantees.
- **Real-world impact:** Such vulnerabilities have been exploited in attacks against RSA, AES, and other cryptographic implementations.

1861
1862
1863
1864
1865
1866
1867
1868
1869
1870

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 language semantics.

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.

1871

C.2.3 EXAMPLE 3

1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889

Example: 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
    }
}
```

```

1890
1891     if (B[i] > 8888) { // Instruction 5
1892         A[i] = A[i] * B[i]; // Instruction 6
1893     }
1894 }

```

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] * B[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
- $n_1 + n_2 + n_3 = 64$

1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997

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

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 illustrates 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:

- it tests understanding of control flow divergence and its impact on performance
- 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 (“Student D”) 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.

1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051

C.2.4 EXAMPLE 4

Example: Combinational vs. Sequential from Verilog

Question:

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

2052 previous value when both a and b are 0. This subtlety makes the design sequential, even without an
 2053 explicit clock.

2054 In the context of computer architecture, understanding such distinctions is critical. Hardware de-
 2055 scription languages (HDLs) like Verilog are used to describe datapaths, control units, and memory
 2056 systems. Misclassifying sequential vs. combinational logic can lead to unintended latches, incorrect
 2057 timing behavior, or hardware inefficiencies. This impacts pipeline stages, critical path delays, and
 2058 ultimately system performance and reliability—fundamental aspects of architecture design.

2059 Analysis of Model Response: The model's response (Student D) incorrectly concluded that the
 2060 code produces a purely combinational circuit with an unintended latch. While it correctly identi ed
 2061 the incomplete case coverage, it failed to reconcile the key fact that incomplete assignments within
 2062 an always @(*) block result in latch inference. This is not “still combinational with a caveat,”
 2063 but a defining feature of sequential logic, since the latch introduces memory behavior.

2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074

2075 C.3 FAILURE MODE 2: ASSUMING UNCONVENTIONAL ARCHITECTURAL PROPERTIES

2076
2077
2078
2079
2080

2081 C.3.1 EXAMPLE 1

2082
2083
2084
2085
2086

2087 Example: Big-Endian Ordering

2089 Context:

2090 Suppose a virtual memory system employs a 64KB page size and a two-level hierarchical
 2091 page table. The physical address of the base of the Level-1 page table (0x01000) is stored
 2092 in a dedicated register called the Page Table Base Register. Both the virtual and physical
 2093 addresses are 20 bits wide. The figure below illustrates the page table structure and the
 2094 breakdown of a virtual address in this system.

2095 Each page table entry (PTE) in both Level-1 and Level-2 tables occupies 4 bytes, and the
 2096 memory is byte-addressable. All pages and page tables are assumed to reside in main
 2097 memory. Each entry in the Level-1 page table contains the physical base address of a
 2098 Level-2 page table, while each entry in a Level-2 page table holds the PTE corresponding
 2099 to a data page. (The diagram is not drawn to scale.) As shown, the L1 index and L2 index
 2100 are used to select the appropriate 4-byte entry in the Level-1 and Level-2 page tables,
 2101 respectively.

2102 A Level-2 page table entry can be decomposed into the following fields (status bits can be
 2103 ignored for this discussion).
 2104
 2105

2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159

Question:

What is the total size of memory required to store both the level 1 and 2 page tables?

Correct Solution:

L1: 4 lines each 4B ! 16B

L2: 4 * 4 B = 64B ! Total = 16 + 64 = 80B

Incorrect Frontier Model Response:

To determine the total memory required for both levels of page tables, we analyze the virtual address structure and calculate the number of entries in each level.

Step 1: Analyze the Virtual Address Breakdown

From the provided diagram, the 20-bit virtual address is divided as follows:

- Bits 19–18: L1 index (p) – 2 bits
- Bits 17–16: L2 index (p) – 2 bits
- Bits 15–0: Page offset – 16 bits (corresponding to the 64KB page size of 2¹⁶ bytes)

Step 2: Calculate Level 1 Page Table Size

- Number of entries in Level 1 = $2^{\text{number of L1 index bits}} = 2^2 = 4$
- Each entry size = 4 bytes
- Total Level 1 size = 4 * 4 = 16 bytes

Step 3: Calculate Level 2 Page Tables Size

- Number of Level 2 page tables = Number of entries in Level 1 = 4
- Number of entries in each Level 2 page table = $2^{\text{number of L2 index bits}} = 2^2 = 4$
- Each entry size = 4 bytes
- Size of one Level 2 page table = 4 * 4 = 16 bytes
- Total Level 2 size = 4 * 16 = 64 bytes

Step 4: Calculate Total Size

$$\begin{aligned} \text{Total memory required} &= \text{Level 1 size} + \text{Total Level 2 size} \\ &= 16 \text{ bytes} + 64 \text{ bytes} = 80 \text{ bytes} \end{aligned}$$

Therefore, the total size of memory required to store both the Level 1 and Level 2 page tables is 80 bytes.

2160 Question Summary and Relevance: This question asks for the memory size required to store
 2161 a two-level hierarchical page table in a 20-bit virtual memory system with 64KB pages. Since
 2162 the system is byte-addressable, the calculation should follow conventional little-endian, byte-level
 2163 memory assumptions common to modern processors. Correct reasoning requires straightforward
 2164 indexing into the page tables and summing their storage requirements.

2165 Analysis of Model Response: The model incorrectly assumes big-endian ordering in the inter-
 2166 pretation of address breakdown. Instead of aligning the lower-order bits with the page offset, the
 2167 response reverses the significance of address portions, misplacing table indices. While big-endian
 2168 addressing is technically valid, it contradicts the universally assumed little-endian default in mod-
 2169 ern architecture, leading to a miscalculated total size. This illustrates how models, absent explicit
 2170 prompt guidance, may diverge into less conventional assumptions not used in real-world practice.
 2171

2172 C.3.2 EXAMPLE 2

2173 Example: Word-Level vs. Byte-Level Addressing

2174 Context:

2175 In this question, you will be given a task and two code snippets in MIPS assembly language.
 2176 You will have to decide which of the code snippets can be used for the task.
 2177

2178 For all the questions assume the following initial values:

2179 Registers:

| Register | Value |
|----------|------------|
| \$s0 | 0x000000FF |
| \$s1 | 0x00000004 |
| \$s2 | 0x00000008 |
| \$s3 | 0x0000000C |

2180 Memory:

| Address | Value |
|------------|------------|
| 0x00000000 | 0x0000FF00 |
| 0x00000004 | 0x000000FF |
| 0x00000008 | 0xFFFFFFFF |
| 0x0000000C | 0x12345678 |

2181 Question:

2182 Write all zeroes to memory starting at 0x0000 4000 for 1024 consecutive memory locations
 2183 (up to 0x0000 5000)
 2184

2185 (A)

```
2186 addi $s0, $s0, 0x1000
2187 LOOP:
2188     sw    $0, 0x4000($s0)
2189     addi $s0, $s0, 1
2190     bne  $s0, $0, LOOP
```

2191 (B)

```
2192 addi $s0, $s0, 0x4000
2193 addi $s1, $s0, 0x1000
2194 addi $s2, $0, 1
2195 LOOP:
2196     sw    $0, 0($s0)
2197     sub  $s1, $s1, $s2
2198     bne  $s0, $s1, LOOP
```

2199 Correct Solution:

2200 A Since the assignment is on register \$s0, which is constant at 0x4000, if the line were to
 2201 read:
 2202

2214
2215 LOOP: sw \$0, 0(\$s1)
2216 it would be correct.
2217
2218 Incorrect Frontier Model Response:
2219 Looking at this problem, I need to analyze which code snippet correctly writes zeroes to
2220 1024 consecutive memory locations starting at 0x00004000.
2221
2222 Let me trace through each code snippet:
2223
2224 Analysis of Code Snippet (A):
2225 Initial state: \$s0 = 0x000000FF
2226 addi \$s0, \$s0, 0x1000 # \$s0 = 0x000000FF+0x1000 = 0x000010FF
2227 LOOP:
2228 sw \$0, 0x4000(\$s0) # Write 0 to address (0x4000 + \$s0)
2229 addi \$s0, \$s0, 1 # Decrement \$s0 by 1
2230 bne \$s0, \$0, LOOP # Continue until \$s0 = 0
2231
2232 This will write to addresses:
2233 • First iteration: 0x4000 + 0x10FF = 0x50FF
2234 • Second iteration: 0x4000 + 0x10FE = 0x50FE
2235 • ...
2236 • Last iteration: 0x4000 + 0x0000 = 0x4000
2237
2238 Number of iterations: 0x10FF + 1 = 4352
2239 Memory range: 0x4000 to 0x50FF (consecutive bytes, going backwards).
2240
2241 Analysis of Code Snippet (B):
2242 Initial state: \$s0 = 0x000000FF, \$s1 = 0x00000004
2243 addi \$s0, \$s0, 0x4000 # \$s0 = 0x000000FF+0x4000 = 0x000040FF
2244 addi \$s1, \$s0, 0x1000 # \$s1 = 0x000040FF+0x1000 = 0x000050FF
2245 addi \$s2, \$0, 1 # \$s2 = 1
2246 LOOP:
2247 sw \$0, 0(\$s0) # Write 0 to address \$s0
2248 sub \$s1, \$s1, \$s2 # \$s1 = \$s1 - 1
2249 bne \$s0, \$s1, LOOP # Continue until \$s0 = \$s1
2250
2251 Critical issue: \$s0 never changes in the loop! It remains 0x000040FF throughout.
2252
2253 This will:
2254 • Always write to the same address: 0x000040FF
2255 • Loop 0x000050FF - 0x000040FF = 0x1000 = 4096 times
2256 • Write the same memory location 4096 times
2257
2258 Target requirement: Write zeroes to 1024 consecutive memory locations from
2259 0x00004000 to 0x00004FFC (1024 words 4 bytes = 4096 bytes total).
2260 Code Snippet (A):
2261 • Writes to wrong address range (0x4000 to 0x50FF instead of 0x4000 to
2262 0x4FFC)
2263 • Writes to individual bytes instead of word boundaries
2264 • Goes backward instead of forward
2265 • At least covers the target memory range (and more)
2266 Code Snippet (B):
2267 • Writes to only one memory location (0x000040FF)

2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321

- This location is outside the target range
- Completely fails to meet the requirement

Neither code snippet correctly implements the required task. However, if I must choose between them:

- Code Snippet (A) is closer to being correct because:
 1. It writes to a memory range that includes the target addresses
 2. It writes zeros to memory (even if to too many locations)
 3. Code Snippet (B) completely fails by writing to only one wrong address

Answer: (A) — Though flawed, it's the only option that writes to multiple memory locations in the vicinity of the target range.

Question Summary and Relevance: The task required writing zeroes to memory locations from 0x00004000 through 0x00005000 – a range of exactly 1024 word-aligned addresses. This detail is crucial because MIPS sw instructions demand word alignment (multiples of 4), and the iteration step must move in increments of 4 to ensure correct coverage of 1024 words. Any solution that treats the loop index as a byte counter or ignores alignment will fail to meet the specification, either by writing to invalid addresses, writing too many times, or missing required locations.

Analysis of Model Response: The model concludes that snippet (A) is “closer to being correct,” overlooking its fundamental flaw: it decrements \$s0 by one each iteration, producing byte-level addressing instead of word-level addressing. This causes misaligned sw operations at addresses such as 0x50FF, 0x50FE, etc., which would raise exceptions on a real MIPS processor. Although Student D recognizes that snippet (B) only writes to a single location, their preference for snippet (A) ignores the fact that (A) still does not meet the requirement – it overshoots the memory range, writes more than 1024 times, and uses byte steps instead of 4-byte steps. The key error in Student D's analysis is failing to recognize that word-level vs. byte-level addressing is a decisive factor that invalidates snippet (A).

C.3.3 EXAMPLE 3

Example: Non-Standard Bit Indexing Patterns

Context:

Let A and B be two unknown 8-bit 2's complement numbers. We know the results of A + B and A ^ B as shown below:

$$A + B = 00110100$$

$$A \oplus B = 11001001$$

Question:

What is the sum A + B expressed in the 8-bit two's complement notation?

On an x86 system, would the carry flag be set after A + B? What about the overflow flag?

Correct Solution:

$$A + B = 11000110$$

Will carry flag be set? Yes

Will overflow flag be set? No

Incorrect Frontier Model Response:

The question asks for the sum A + B in 8-bit two's complement notation and the state of the carry and overflow flags on an x86 system after this addition.

We are given:

- $A + B = 00110100$ (Let this be P, where $P_i = A_i + B_i$)
- $A \oplus B = 11001001$ (Let this be G, where $G_i = A_i \oplus B_i$)

2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2370
2371
2372
2373
2374
2375

For 8-bit binary addition, the sum bit Sum_i at position i and the carry-out bit $c.out_i$ are determined by the individual bits A_i, B_i , and the carry-in bit $c.in_i$. The relations are:

- $Sum_i = A_i \oplus B_i \oplus c.in_i$
- $c.out_i = (A_i \wedge B_i) \vee (A_i \wedge c.in_i) \vee (B_i \wedge c.in_i)$

We can simplify these using P and G :

- $Sum_i = P_i \oplus c.in_i$
- $c.out_i = G_i \vee (P_i \wedge c.in_i)$

We can trace the carries and sum bits from the least significant bit (LSB, bit 0) to the most significant bit (MSB, bit 7). The initial carry $c.in_0$ is 0.

Step-by-Step Calculation of $A + B$ and Flags:

| Bit i | $P_i = (A \oplus B)_i$ | $G_i = (A \wedge B)_i$ | $c.in_i$ | $Sum_i = P_i \oplus c.in_i$ | $c.out_i = G_i \vee (P_i \wedge c.in_i)$ |
|---------|------------------------|------------------------|----------|-----------------------------|--|
| 0 | 0 | 1 | 0 | $0 \oplus 0 = 0$ | $1 \vee (0 \wedge 0) = 1$ |
| 1 | 0 | 0 | 1 | $0 \oplus 1 = 1$ | $0 \vee (0 \wedge 1) = 0$ |
| 2 | 1 | 0 | 0 | $1 \oplus 0 = 1$ | $0 \vee (1 \wedge 0) = 0$ |
| 3 | 0 | 1 | 0 | $0 \oplus 0 = 0$ | $1 \vee (0 \wedge 0) = 1$ |
| 4 | 1 | 0 | 1 | $1 \oplus 1 = 0$ | $0 \vee (1 \wedge 1) = 1$ |
| 5 | 1 | 0 | 1 | $1 \oplus 1 = 0$ | $0 \vee (1 \wedge 1) = 1$ |
| 6 | 0 | 1 | 1 | $0 \oplus 1 = 1$ | $1 \vee (0 \wedge 1) = 1$ |
| 7 | 0 | 1 | 1 | $0 \oplus 1 = 1$ | $1 \vee (0 \wedge 1) = 1$ |

From the table:

The sum $A + B$ (represented by $Sum_7 :: Sum_0$) is 11000110 in 8-bit binary.

1. Sum $A + B$ expressed in 8-bit two's complement notation:
 $A + B = 11000110$

To convert this to decimal: since the MSB is 1, it is negative.
 Two's complement of 11000110: invert !00111001, add 1 !00111010.
 Decimal value = $32 + 16 + 8 + 2 = 58$.
 Therefore, 11000110 represents -58.

2. Carry Flag (CF) after $A + B$ on x86:

The Carry Flag is set if there is a carry-out from the MSB. From the table, $c.out_7 = 1$.
 Therefore, CF = 1 (set).

3. Overflow Flag (OF) after $A + B$ on x86:

Overflow is set if $c.in_7 \oplus c.out_7 = 1$. From the table, $c.in_7 = 1$, $c.out_7 = 1$, so
 $OF = 1 \oplus 1 = 0$.

Therefore, OF = 0 (not set).

Final Answer:

- Sum $A + B$: 11000110 (-58 in decimal)
- Carry Flag: Set (CF = 1)
- Overflow Flag: Not set (OF = 0)

Question Summary and Relevance: The question under consideration asks students to compute the sum of two unknown 8-bit two's complement numbers, A and B , given only their bitwise XOR ($A \oplus B$) and AND ($A \wedge B$) results. Students must then express $A + B$ in 8-bit two's complement notation and determine the states of the carry and overflow flags on an x86 system. This problem is a classic test of digital arithmetic reasoning, requiring careful use of propagator and generate (P and G) signals, correct bit-by-bit carry calculation, and strict adherence to conventional bit indexing (LSB =

2376 bit 0 through MSB = bit 7). It highlights the importance of consistent bit ordering when performing
 2377 ag analysis, since arithmetic correctness and ag determination depend on accurate propagation of
 2378 carries from least significant to most significant bits.

2379 Analysis of Model Response: The model response approach reveals a misunderstanding rooted in
 2380 non-standard bit indexing patterns. While they correctly introduced propagate and generate de-
 2381 nitions, their subsequent tracing of carries and sums implicitly reversed or misapplied the conventional
 2382 numbering scheme. By failing to consistently treat the least significant bit as index 0 and the most
 2383 significant bit as index 7, the student introduced misalignment between bit positions and the carries
 2384 they computed. This mistake corrupted both the intermediate arithmetic reasoning and the interpre-
 2385 tation of overflow and carry-ags. The response therefore illustrates a key failure mode: even with
 2386 correct formulas, adopting an unconventional or inconsistent bit indexing convention undermines
 2387 the entire analysis, leading to incorrect conclusions about the final result and ag states.

2389 C.4 FAILURE MODE 3: MODELING AND TRACKING SYSTEM STATE

2390 C.4.1 EXAMPLE 1

2392 Example: Test-and-Set States

2394 Context:

2395 You are writing a queue to be used in a multi-producer/single-consumer application. (Pro-
 2396 ducer threads write messages that are read by one consumer.) We assume here a queue with
 2397 finite space. The basic code is shown below.

2398 TST rs, Imm(rt) is the test-and-set instruction, which atomically loads the value at Imm(rt)
 2399 into rs, and if the value is zero, updates the memory location at Imm(rt) to 1. This atomic
 2400 instruction is useful for implementing locks: a value of 1 at the memory location indicates
 2401 that someone holds the lock, and a value of 0 means the lock is free.

2402 Producer pushes a message onto queue: (memory operations in bold)

```
2403 void push(int ** tail_ptr, int * tail_write_lock, int message) {
2404     while (lock_try(tail_write_lock) == false);
2405     ** tail_ptr = message;
2406     * tail_ptr++;
2407     lock_release(tail_write_lock);
2408 }
2409
```

```
2410 # R1 - contains address of data to enqueue
2411 # R2 - contains the address of the tail pointer of queue
2412 # R3 - address of tail pointer write lock
2413 P1 SpinLock:TST R4, 0(R3) # try to acquire tail write lock
2414 P2 BNEZ R4, R4, SpinLock
2415 P3 LD R4, 0(R2) # get tail pointer
2416 P4 ST R1, 0(R4) # write message to tail
2417 P5 ADD R4, R4, 4 # update tail pointer
2418 P6 ST R4, 0(R2)
2419 P7 ST R0, 0(R3) # release lock
2420
```

```
2421 int pop(int ** head_ptr, int ** tail_ptr) {
2422     while ( * head_ptr == * tail_ptr);
2423     int message = ** head_ptr;
2424     * head_ptr++;
2425     return message;
2426 }
2427
```

```
2428 # R1 - will receive address contained in message
2429 # R2 - contains the address of the head pointer of queue
```

2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2480
2481
2482
2483

```
# 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?

2484
2485
2486
2487
2488
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2530
2531
2532
2533
2534
2535
2536
2537

Correct Solution:

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 tailptr: - 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 newtail: - 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 tailptr: - 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 tailptr): - 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 newtail): - 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 tailptr): - 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 ! 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 ! M (1 transition) - Load tail pointer: I ! S/E (1 transition) - Write message: S/E ! M (1 transition) - Update tail pointer: S/E ! M (1 transition) - Release lock: M ! I (1 transition)

2. Consumer: - Load tail pointer: I ! S (1 transition) - Invalidate producer's tail pointer: M ! 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 newtail | 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

Example: Identifying Data Dependencies from Instruction Trade

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

```

2592
2593
2594
2595
2596
2597
2598
2599
2600
2601
2602
2603
2604
2605
2606
2607
2608
2609
2610
2611
2612
2613
2614
2615
2616
2617
2618
2619
2620
2621
2622
2623
2624
2625
2626
2627
2628
2629
2630
2631
2632
2633
2634
2635
2636
2637
2638
2639
2640
2641
2642
2643
2644
2645

```
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 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. Correctly solving it requires carefully tracking data flow dependencies across instructions and determining when a later instruction consumes a value produced by an earlier one thus creating a stall in the processor pipeline.

2646 Analysis of Model Response: The model successfully tracked register states but failed to capture
 2647 memory updates, resulting in an incomplete representation of the system state. This limitation led
 2648 to errors in identifying Read-after-Write (RAW) dependencies and ultimately produced an incor-
 2649 rect answer, illustrating the model's weakness in handling memory-based interactions and implicit
 2650 architectural information.

2651

2652 C.5 FAILURE MODE 4: SENSITIVITY TO QA MODALITY

2653

2654 C.5.1 EXAMPLE 1

2655

2656

2657

2658

2659

2660

2661

In the example below, the model failed to properly interpret the RAT (Register Alias Table) and reservation station mappings shown in the diagram, particularly overlooking the source tags and sequential ordering information. This caused it to incorrectly mark valid destination registers as unknown. However, the diagram provided sufficient information for complete mapping if the model had properly cross-referenced the source tags with RAT entries and considered the program execution order across processing units.

2662

Example: Filling in Instruction Fields From Out-of-Order Execution Snapshot

2663

2664

2665

2666

2667

2668

2669

2670

2671

2672

2673

2674

2675

2676

2677

2678

2679

2680

2681

2682

2683

2684

2685

2686

2687

2688

2689

2690

2691

2692

2693

2694

2695

2696

2697

2698

2699

Context:

The diagram below shows a snapshot at a particular point in time of various parts (reservation stations and register alias table) of the microarchitecture for an implementation supporting out-of-order execution in the spirit of Tomasulo's Algorithm. Note that there is an adder and a multiplier in this machine. The processor is supplied with a seven instruction program following reset. The state below was captured at some point in time during the execution of these seven instructions. Anything marked with a - is unknown and can't be relied upon for your answer. You should assume that the bottommost instruction in the reservation station arrived earliest and the topmost instruction in the reservation station arrived last.

Question:

Fill in the instruction opcodes, source, and destination registers in the table below.

Correct Solution:

| OP | DEST | SRC1 | SRC2 |
|-----|------|------|------|
| ADD | R3 | R1 | R2 |

2700
2701
2702
2703
2704
2705
2706
2707
2708
2709
2710
2711
2712
2713
2714
2715
2716
2717
2718
2719
2720
2721
2722
2723
2724
2725
2726
2727
2728
2729
2730
2731
2732
2733
2734
2735
2736
2737
2738
2739
2740
2741
2742
2743
2744
2745
2746
2747
2748
2749
2750
2751
2752
2753

```
ADD R1 R2 R3
MUL R4 R1 R1
MUL R5 R2 R1
ADD R2 R2 R5
MUL R5 R4 R1
ADD R5 R2 R5
```

Incorrect Frontier Model Response:
Answer to Out-of-Order Execution Instruction Reconstruction

To solve this problem, I need to analyze the reservation stations and Register Alias Table (RAT) to determine what instructions are being executed. I'll examine the dependencies between instructions and map the data flow.

Analysis of the RAT

From the RAT, I can determine which reservation stations are producing values for specific registers:

- R1: Waiting for result from RS A (Tag = A)
- R2: Waiting for result from RS B (Tag = B)
- R4: Waiting for result from RS C (Tag = C)
- R5: Waiting for result from RS D (Tag = D)

Analysis of Reservation Stations

Addition Reservation Stations:

- 1) RS A (earliest): Both sources are valid with values 20 and 30. Will produce value for R1. Operation must be ADD R1, ?, ?; since both values are ready: ADD R1, #20, #30 (or using registers with those values).
- 2) RS B: SRC1 valid with value 20; SRC2 waiting for RS F. Will produce value for R2. Operation must be ADD R2, #20, ?.
- 3) RS D: SRC1 waiting for RS B (which produces R2); SRC2 waiting for RS E. Will produce value for R5. Operation must be ADD R5, R2, ?.

Multiplication Reservation Stations:

- 1) RS C (earliest): Both sources waiting for RS A (which produces R1). Will produce value for R4. Must be using the same value twice: MUL R4, R1, R1.
- 2) RS F: SRC1 waiting for RS A (which produces R1); SRC2 valid with value 20. Destination register unknown. Operation must be MUL ?, R1, #20.
- 3) RS E: SRC1 waiting for RS C (which produces R4); SRC2 waiting for RS A (which produces R1). Destination register unknown. Operation must be MUL ?, R4, R1.

Completed Table

Reservation Station A: Opcode = ADD, Source1 = #20, Source2 = #30, Destination = R1
 Reservation Station B: Opcode = ADD, Source1 = #20, Source2 = ?, Destination = R2
 Reservation Station C: Opcode = MUL, Source1 = R1, Source2 = R1, Destination = R4
 Reservation Station D: Opcode = ADD, Source1 = R2, Source2 = ?, Destination = R5
 Reservation Station E: Opcode = MUL, Source1 = R4, Source2 = R1, Destination = ?
 Reservation Station F: Opcode = MUL, Source1 = R1, Source2 = #20, Destination = ?

Note: For the unknown destination registers (marked with ?), I cannot determine their values from the provided information, as they are not shown in the RAT excerpt.

2754 C.5.2 EXAMPLE 2

2755

2756

Example: Deadlock-Free Min-Cut

2757

Question:

2758

2759

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?

2760

2761

2762

2763

2764

2765

2766

2767

2768

2769

2770

2771

2772

2773

2774

- ZW ! WX; ZY ! YX
- ZW ! WX; ZY ! YX; YW ! WX; ZY ! YW
- YW ! WX; ZY ! YW; XY ! YZ; XW ! WZ
- ZW ! WX

2775

2776

2777

2778

2779

Correct Solution:

2780

2781

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.

2782

2783

2784

Incorrect Frontier Model Response:

2785

2786

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

2787

2788

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

2789

2790

2791

2792

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

2793

2794

2795

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.

2796

2797

2798

2799

2800

2801

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

2802

2803

2804

2805

2806

2807

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.

2808
2809
2810
2811
2812
2813
2814
2815
2816
2817
2818
2819
2820
2821
2822
2823
2824
2825
2826
2827
2828
2829
2830
2831
2832
2833
2834
2835
2836
2837
2838
2839
2840
2841
2842
2843
2844
2845
2846
2847
2848
2849
2850
2851
2852
2853
2854
2855
2856
2857
2858
2859
2860
2861

- 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.

The question tests the models' understanding of deadlock freedom conditions. 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.

C.6 SUCCESSFUL MODEL RESPONSE

In the example below, we illustrate a scenario where a model successfully generates the correct answer to a question in the dataset. Although the responses

the frontier model's response aligns conceptually with the answer key by addressing the same advantages using more verbose phrasing. Their 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, their 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.

Example: 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.

2862
2863
2864
2865
2866
2867
2868
2869
2870
2871
2872
2873
2874
2875
2876
2877
2878
2879
2880
2881
2882
2883
2884
2885
2886
2887
2888
2889
2890
2891
2892
2893
2894
2895
2896
2897
2898
2899
2900
2901
2902
2903
2904
2905
2906
2907
2908
2909
2910
2911
2912
2913
2914
2915

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.

C.7 EXAMPLE QUESTIONS FOR CASE STUDY

This section presents concrete questions from QuArch that are similar to the types of questions asked in the memory controller design case study (Section 4.4). Please refer to Appendix D.11 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 ANALYZING MEMORY TRACES

Example: 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

Example: 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.

2916
2917
2918
2919
2920
2921
2922
2923
2924
2925
2926
2927
2928
2929
2930
2931
2932
2933
2934
2935
2936
2937
2938
2939
2940
2941
2942
2943
2944
2945
2946
2947
2948
2949
2950
2951
2952
2953
2954
2955
2956
2957
2958
2959
2960
2961
2962
2963
2964
2965
2966
2967
2968
2969

* 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.

2970
2971
2972
2973
2974
2975
2976
2977
2978
2979
2980
2981
2982
2983
2984
2985
2986
2987

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

2988
2989
2990

C.7.2 MEMORY CONTROLLER CONFIGURATIONS

2991
2992
2993
2994
2995
2996
2997
2998
2999
3000
3001
3002
3003
3004
3005
3006
3007

Example: 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.

3008
3009
3010

Example: DRAM Heirarchy Configuration

3011
3012
3013
3014
3015
3016
3017
3018
3019
3020
3021
3022
3023

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

3024
3025
3026
3027
3028
3029
3030
3031
3032
3033
3034
3035
3036
3037
3038
3039
3040
3041
3042
3043
3044
3045
3046
3047
3048
3049
3050
3051
3052
3053
3054
3055
3056
3057
3058
3059
3060
3061
3062
3063
3064
3065
3066
3067
3068
3069
3070
3071
3072
3073
3074
3075
3076
3077

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.

Example: 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 $100R * 5ns * 2ranks * 8banks / 64ms = 1$ $R = 800,000$ We find that with 800k rows per bank, the command bus utilization reaches 100

D PROMPT TEMPLATES

D.1 LLM PROMPT FOR STUDENT LLM

Student LLM 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:

$\hat{f}_{context}g$

$\hat{f}_{context_images_placeholder}g$

Question:

$\hat{f}_{question}g$

3078
 3079
 3080
 3081
 3082
 3083
 3084
 3085
 3086
 3087
 3088
 3089
 3090
 3091
 3092
 3093
 3094
 3095
 3096
 3097
 3098
 3099
 3100
 3101
 3102
 3103
 3104
 3105
 3106
 3107
 3108
 3109
 3110
 3111
 3112
 3113
 3114
 3115
 3116
 3117
 3118
 3119
 3120
 3121
 3122
 3123
 3124
 3125
 3126
 3127
 3128
 3129
 3130
 3131

Please provide your answer:

This prompt is used to simulate the “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.2 LLM PROMPT FOR MCQ RESPONSE

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:

$\hat{f}_{question}g$

Choices:

$\hat{f}_{choices}g$

Please conclude your response with a JSON object containing your final answer. The JSON object must match this schema: \hat{f}_{final_answer} : “ h A, B, C, or D” \hat{f}_{g} .

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.3 LLM PROMPT FOR FRQ LLM-AS-A-JUDGE

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.

3132
3133
3134
3135
3136
3137
3138
3139
3140
3141
3142
3143
3144
3145
3146
3147
3148
3149
3150
3151
3152
3153
3154
3155
3156
3157
3158
3159
3160
3161
3162
3163
3164
3165
3166
3167
3168
3169
3170
3171
3172
3173
3174
3175
3176
3177
3178
3179
3180
3181
3182
3183
3184
3185

Question Context:

$\tilde{f}_{\text{context}}g$

$\tilde{f}_{\text{context_images_placeholder}}g$

Question:

$\tilde{f}_{\text{question}}g$

Correct Solution:

$\tilde{f}_{\text{solution}}g$

$\tilde{f}_{\text{solution_images_placeholder}}g$

Student’s Answer:

$\tilde{f}_{\text{student_answer}}g$

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 answers 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”.

3186
3187
3188
3189
3190
3191
3192
3193
3194
3195
3196
3197
3198
3199
3200
3201
3202
3203
3204
3205
3206
3207
3208
3209
3210
3211
3212
3213
3214
3215
3216
3217
3218
3219
3220
3221
3222
3223
3224
3225
3226
3227
3228
3229
3230
3231
3232
3233
3234
3235
3236
3237
3238
3239

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:

\bar{r} question g

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: \bar{r} question g

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 have

3240
3241
3242
3243
3244
3245
3246
3247
3248
3249
3250
3251
3252
3253
3254
3255
3256
3257
3258
3259
3260
3261
3262
3263
3264
3265
3266
3267
3268
3269
3270
3271
3272
3273
3274
3275
3276
3277
3278
3279
3280
3281
3282
3283
3284
3285
3286
3287
3288
3289
3290
3291
3292
3293

to provide your response should be in the format without any additional text or explanation:

```
[f“best_selection”: “[BEST CATEGORY HERE]”,
“justification”: “[JUSTIFICATION #1 HERE]” g,

f“second_best_selection”: “[SECOND BEST CATEGORY HERE]”,
“justification”: “[JUSTIFICATION #2 HERE]” g]
```

D.6 LLM PROMPT FOR 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:
“*f*excerpt*g*”

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 ns
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:

```
[
f
“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”: “fspecified_answer_choice[‘Cloze’]g”,
“context”: “[JUSTIFICATION HERE AS CONTEXT]”,
“type”: “Cloze”
g
]
```

3294 D.7 LLM PROMPTS FOR MCQ FILTERING
3295
3296
3297

MCQ Filtering Prompt 1

3298 You are a computer architecture expert.
3299

3300 You have been asked the following question:

3301 \bar{f} question g
3302

3303 Here are the options:

3304 \bar{f} options g
3305

3306 As a computer architecture expert, would you need additional context to correctly answer
3307 the question?

3308
3309 Please answer with one word: "YES" or "NO".

3310 Then provide one short sentence to justify your answer reasoning. Return your response in
3311 this exact format with zero other characters for formatting before or after:

3312 \bar{f}

3313 "answer": "[ANSWER HERE: YES/NO]",

3314 "justification": "[JUSTIFICATION HERE]"

3315 g
3316
3317

MCQ Filtering Prompt 2

3320 You are a computer architecture expert.
3321

3322 You have been asked the following question:

3323 \bar{f} question g
3324

3325 Here are the options:

3326 \bar{f} options g
3327

3328 As a computer architecture expert, of the provided options is there *only one* answer that
3329 is correct?

3330
3331 Please answer with one word: "YES" or "NO".

3332 Then provide one short sentence to justify your answer reasoning. Return your response in
3333 this exact format with zero other characters for formatting before or after:

3334 \bar{f}

3335 "answer": "[ANSWER HERE: YES/NO]",

3336 "justification": "[JUSTIFICATION HERE]"

3337 g
3338
3339

MCQ Filtering Prompt 3

3342 You are a new graduate student reading about computer architecture to learn the subject.
3343

3344 You have been given the following quote from a computer architecture excerpt to read for
3345 context:

3346 \bar{f} context g
3347

3348
3349
3350
3351
3352
3353
3354
3355
3356
3357
3358
3359
3360
3361
3362
3363
3364
3365
3366
3367
3368
3369
3370
3371
3372
3373
3374
3375
3376
3377
3378
3379
3380
3381
3382
3383
3384
3385
3386
3387
3388
3389
3390
3391
3392
3393
3394
3395
3396
3397
3398
3399
3400
3401

You have now been asked the following question:

\bar{f} question g

Here are the options:

\bar{f} options g

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:

\bar{f}

“answer”: “[ANSWER HERE: YES/NO]”,

“justification”: “[JUSTIFICATION HERE]”

g

MCQ Filtering Prompt 4

You are a computer architecture expert.

You have been given the following quote from a computer architecture excerpt for context:

\bar{f} context g

You have now been asked the following question:

\bar{f} question g

Here are the options:

\bar{f} options g

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:

\bar{f}

“answer”: “[ANSWER HERE: LETTER OF OPTION]”,

“justification”: “[JUSTIFICATION HERE]”

g

D.8 LLM PROMPT FOR EXAM QUESTION TEXT EXTRACTION

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

3402
3403
3404
3405
3406
3407
3408
3409
3410
3411
3412
3413
3414
3415
3416
3417
3418
3419
3420
3421
3422
3423
3424
3425
3426
3427
3428
3429
3430
3431
3432
3433
3434
3435
3436
3437
3438
3439
3440
3441
3442
3443
3444
3445
3446
3447
3448
3449
3450
3451
3452
3453
3454
3455

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.

This is the prompt we use to extract the text for each question in our crowdsourced exam dataset. 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.

3456 D.9 LLM PROMPT FOR EXAM QUESTION IMAGE EXTRACTION
 3457
 3458
 3459
 3460

Exam Question Image Extraction Prompt

3461 You are a language model assisting with the digitization of academic exam content in Com-
 3462 puter Architecture.

3463 **Input:**

3464 You are provided with:

- 3465 1) A PDF file containing one problem of an exam. If part of another problem is included,
 3466 ignore this and only focus on the current pdf file.
 3467 2) A .json-styled txt file containing the problem’s extracted text. It may contain
 3468 some or all of the following empty-list fields: ‘ ‘ probl em_context_fi gures”,
 3469 ‘ ‘ subprobl em_context_fi gures”, ‘ ‘ subprobl em_sol uti on_fi gures”,
 3470 and ‘ ‘ probl em_sol uti on_fi gures” .
 3471 3) PNG images containing tables, diagrams, circuit schematics, or block diagrams that may
 3472 or may not pertain to this problem. The names of the images provided are as follows, in
 3473 order: *fimagesg*.

3474 The given problem may be a standalone problem or consist of multiple sub-problems. Each
 3475 problem or sub-problem may contain:

- 3476 – Main context figures which are necessary to understanding the main problem, or ALL of
 3477 the sub-problems.
 3478 – Sub-problem-specific figures that are separate from both the main problem context and the
 3479 other sub-problems.
 3480 – Solutions to the main problem, which may be typed or handwritten.
 3481 – Solutions to a certain sub-problem, which may be typed or handwritten.

3482 **Your task:**

3483 Match each image file name (table, diagram, circuit schematic, or block diagram) to its cor-
 3484 rect association in the exam. Each image should have one of the four possible associations:

- 3485 – The main problem question (‘ ‘ probl em_context_fi gures”)
 3486 – The sub-problem question (‘ ‘ subprobl em_context_fi gures”)
 3487 – The main problem solution (‘ ‘ probl em_sol uti on_fi gures”)
 3488 – The sub-problem solution (‘ ‘ subprobl em_sol uti on_fi gures”)

3489 At times, the context or question in the main problem/sub-problem question/solution will
 3490 include phrases (such as “The table below” or “The following diagram”) that indicate a
 3491 visual image falls under that category.

3492 Your output should be a modified version of the given JSON, but with each of the figure
 3493 fields populated with lists of the relevant image names (files ending in .png, .jpg, etc.), for
 3494 example:

3495 ‘ ‘ probl em_context_fi gures”: [‘ ‘ i mage_name. png”,
 3496 ‘ ‘ i mage_name_2. png”]

3497 No content, descriptions, or recreations of the image should be included in the output; only
 3498 the file name should be included. If a provided image is not part of the current PDF file and
 3499 instead is part of other problem(s), omit it. If the provided JSON includes image file names
 3500 that are not present in the images provided, omit those too.

3501 Be as precise as possible in your associations. Only include the dictionary; do not include
 3502 reasoning. If none of the images given pertain to the problem, just output the original JSON
 3503 given.

3504
 3505
 3506 This is the prompt we use to associate figures with their correct roles in the digitized exam prob-
 3507 lems. By structuring images into categories such as problem context, subproblem context, problem
 3508 solution, and subproblem solution, we can align visual information with text-based question content.
 3509 This ensures that diagrams, tables, and schematics are consistently linked to their intended question
 or solution, enabling precise and reproducible dataset construction.

3510 D.10 LLM PROMPT FOR EXAM QUESTION VERIFICATION
 3511

3512 Exam Question Verification
 3513

3514 You are a reader tasked with verifying the accuracy of an automated document parser. A
 3515 computer architecture exam PDF has been fed through the parser to produce standalone
 3516 question-answer pairs in a certain JSON format, and you must compare the parsed questions
 3517 to the original questions in the PDF.

3518 **Input:** You are given (1) the original PDF of $\widehat{f}_{\text{problem_name}g}$ of the exam which may be
 3519 broken down into subproblems, (2) the JSON dictionary produced by the parser, and (3)
 3520 images that the parser has deemed are associated with the question and/or its subproblems.
 3521 The names of the images provided are as follows, in order: $\widehat{f}_{\text{images}g}$. Your task is to
 3522 determine if the parser has correctly extracted the text and images while staying true to the
 3523 original PDF.

3524 **A problem is correct only if all of the following are satisfied:**

- 3525 a) The extracted problem text is nearly identical, word-for-word, to the original PDF's text
 3526 (special characters/math symbols may appear as unicode-escaped or equivalent).
 3527 b) The problem's "question" and "solution" fields are both populated. Excep-
 3528 tion: if the original solution is purely an image, that image must be correctly associated in
 3529 "solution_figures".
 3530 c) The problem is standalone: the "context" and "context_figures" provide all
 3531 information needed, even if the original referenced prior problems.
 3532 d) No part of the solution is revealed in "context" or "context_figures".
 3533 e) All images are extracted/cropped correctly and categorized correctly as context vs. solu-
 3534 tion figures.
 3535 f) All tables are extracted correctly. If parsed as text instead of image, the table must be
 3536 recreatable from the extracted text and usable to answer the question.
 3537 g) For fill-in-the-blank or fill-in-the-chart, the blank version must be provided in the ques-
 3538 tion/context or "context_figures".

3539 **Output format:**

3540 Return a modified version of the given JSON dictionary. Set each
 3541 "passed_llm_verification" field to true or false (unquoted). For every
 3542 item marked false, add a "reasoning" field explaining which conditions failed.
 3543 Return exactly one dictionary and nothing else.

3544 **Conservatism:** Err on the side of false. Avoid false positives; false negatives are acceptable.

3545 This prompt is used to audit the parser's extracted question-answer pairs against the original exam
 3546 source. It enforces strict, itemized criteria for textual fidelity, self-contained context, correct im-
 3547 age/table extraction, and proper figure categorization, and it standardizes the verification output by
 3548 toggling passed_llm_verification and adding concise reasoning where failures are de-
 3549 tected.

3550 D.11 LLM PROMPT FOR MEMORY CONTROLLER DESIGN CASE STUDY
 3551

3552 Exam Question Verification
 3553

3554 You are a computer architecture expert specializing in DRAM memory controller design.
 3555 You have been tasked with tuning the design of a custom DRAM memory controller to
 3556 optimize for ****POWER**** (lower is better).

3557 **=== SEARCH BUDGET ===**

3558 You are on iteration current_iteration of 40.

3559 Consider this when deciding your strategy:

- 3560 - Early iterations: Consider exploring diverse configurations to understand the design space.
 3561 - Later iterations: Consider exploiting promising regions by making smaller refinements to
 3562 the best configurations.
 3563

```

3564
3565 === DRAM CONTROLLER DESIGN PARAMETERS ===
3566 You control 10 DISCRETE design parameters, each represented as an INTEGER INDEX:
3567 0: PagePolicy, Options: ["Open", "OpenAdaptive", "Closed", "ClosedAdaptive"]
3568 1: Scheduler, Options: ["Fifo", "FrFcfsGrp", "FrFcfs"]
3569 2: SchedulerBuffer, Options: ["Bankwise", "ReadWrite", "Shared"]
3570 3: RequestBufferSize, Options: [1, 2, 4, 8, 16, 32, 64, 128]
3571 4: RespQueue, Options: ["Fifo", "Reorder"]
3572 5: RefreshPolicy, Options: ["NoRefresh", "AllBank"]
3573 6: RefreshMaxPostponed, Options: [1,2,4,8]
3574 7: RefreshMaxPulledin, Options: [1,2,4,8]
3575 8: Arbiter, Options: ["Simple", "Fifo", "Reorder"]
3576 9: MaxActiveTransactions, Options: [1, 2, 4, 8, 16, 32, 64, 128]
3577
3578 === CONFIGURATIONS TRIED SO FAR ===
3579 Here are the top 10 best configurations you've found so far, ranked by Power (W) (best
3580 first):
3581 #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,
3582 Latency=0.000114290 s
3583 #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,
3584 Latency=0.085218750 s
3585 #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,
3586 Latency=0.000070795 s
3587 ...
3588 Here are the last 10 configurations you've tried:
3589 #1: [1.0, 2.0, 2.0, 3.0, 1.0, 0.0, 0.0, 0.0, 0.0, 2.0]
3590 #2: [1.0, 2.0, 2.0, 2.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0]
3591 #3: [1.0, 2.0, 2.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
3592 ...
3593 IMPORTANT: You CANNOT propose a configuration that exactly matches any of the Top
3594 10 or Last 10 configs you've tried listed above as proposing a duplicate configuration wastes
3595 an evaluation.
3596
3597 === MEMORY TRACE SAMPLE ===
3598 Below is a sample of the workload's memory trace (format: "timestamp: operation
3599 address"):
3600 — Beginning (lines 0-29 of 10000) —
3601 0: read 0x0
3602 1: read 0x40
3603 2: read 0x80
3604 3: read 0xC0
3605 4: read 0x100
3606 ...
3607 — Middle (lines 4985-5014 of 10000) —
3608 4985: read 0x190BBCC0
3609 4986: read 0x26EEB740
3610 4987: read 0x29CEE6C0
3611 4988: read 0x3771780
3612 4989: read 0x277DC1C0
3613 ...
3614 — End (lines 9970-9999 of 10000) —
3615 9970: read 0x4A880
3616 9971: read 0x4A8C0
3617 9972: read 0x4A900

```

3618
3619
3620
3621
3622
3623
3624
3625
3626
3627
3628
3629
3630
3631
3632
3633
3634
3635
3636
3637
3638
3639
3640
3641
3642
3643
3644
3645
3646
3647
3648
3649
3650
3651
3652
3653
3654
3655
3656
3657
3658
3659
3660
3661
3662
3663
3664
3665
3666
3667
3668
3669
3670
3671

```
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 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)

This prompt is used for the DRAM controller case study (Section 4.4). It tasks the LLM with answering 3 questions and then proposing a new memory controller design configuration. current_i iteration is the i-th round of the 40 evaluation budget provided to the LLM. Thirty lines of the workload's memory trace start, middle, and end is sampled to add to the prompt (ellipses are used for brevity here).