Avaliação de LLMs Multimodais na Modalidade Programação da Olimpíada Brasileira de Informática: Uma Arquitetura Experimental

Victor Hugo S. Ângelo
Instituto da Computação - Universidade Federal de Alagoas
Maceió, Brasil
vhsa@ufal.ic.br

Lucas Heron S. Anchieta
Instituto da Computação - Universidade Federal de Alagoas
Maceió, Brasil
lhsa@ic.ufal.br

Matheus Ryan S. Nascimento
Instituto da Computação - Universidade Federal de Alagoas
Maceió, Brasil
mrysn@ic.ufal.br

Ricardo Vinicius A. Fernandes
Instituto da Computação - Universidade Federal de Alagoas
Maceió, Brasil
rvaf@ic.ufal.br

Evandro de B. Costa
Instituto da Computação - Universidade Federal de Alagoas
Maceió, Brasil
evandro@ic.ufal.br

Resumo

Este artigo tem como objetivo desenvolver e avaliar uma arquitetura para medir a capacidade de Grandes Modelos de Linguagem Multimodais (MLLMs) na resolução de desafios algorítmicos complexos na Modalidade Programação da Olimpíada Brasileira de Informática (OBI). Para isso, foi construída uma infraestrutura automatizada e experimental composta por um agente gerador de código baseado em LLM e um avaliador (Judge) para validação dos casos de teste. Dessa forma, um orquestrador gerencia o fluxo entre o banco de problemas, a estruturação de templates de prompts e os serviços externos (LLMs). Para simplificação dos resultados, adotou-se a seguinte configuração: o código gerado em Python e a técnica de prompt zero-shot, comparando-se enunciados contendo imagens no formato Base64 com os enunciados puramente textuais. O estudo traz um banco de questões ainda não explorado por MLLMs, além de traçar um paralelo pertinente entre a programação competitiva e a Engenharia de Software, no qual a interpretação de enunciados assemelha-se à modelagem de requisitos de sistemas e a validação de corner cases reflete a prática de testes unitários. Os resultados demonstram que o impacto da inclusão de imagens varia significativamente entre as ferramentas; embora o contexto visual possa adicionar ruído ou complexidade em alguns cenários, a análise comparativa revelou uma melhora no desempenho em problemas avançados de grafos e programação dinâmica em 3 dos 5 modelos testados, evidenciando que o suporte multimodal atua como um fator relevante para a resolução correta de desafios de alta complexidade algorítmica.

Palavras-chave

Modelos de Linguagem Multimodais, Geração Automática de Código, Engenharia de Software Experimental, Teste de Software, Programação Competitiva

1 Introdução

A interpretação precisa de requisitos visuais é central na Engenharia de Software [10] e na programação competitiva. Nos problemas da Olimpíada Brasileira de Informática (OBI), ilustrações são frequentemente a chave para a compreensão de restrições algorítmicas, sem as quais a modelagem da solução torna-se incorreta.

Impulsionados pelas leis de escala [6], os Grandes Modelos de Linguagem (LLMs) ganharam notoriedade na geração de código [3] e na resolução de programação competitiva [18] e exames educacionais [9, 11, 14]. Com o avanço do *Visual Instruction Tuning* [7], os Modelos de Linguagem Multimodais (MLLMs) ampliaram essa capacidade cognitiva para contextos com imagens [4], incluindo problemas matemáticos [8], avaliações universitárias [1, 17] e, no ciclo de software, engenharia de requisitos [10], testes [13] e correção de bugs visuais [16].

Apesar desses avanços, o impacto do contexto visual no raciocínio algorítmico de nível educacional inicial permanece pouco explorado. A OBI é uma das principais portas de entrada ao pensamento computacional no Brasil, e seus enunciados dependem fortemente de figuras. Compreender como os MLLMs lidam com essas representações é relevante para a Inteligência Artificial e para a Engenharia de Software Experimental.

Neste trabalho, avaliamos uma arquitetura experimental automatizada para medir a capacidade de MLLMs na resolução de problemas da OBI [12]. A infraestrutura integra um agente gerador e um juiz automático (*Judge*) para validação com casos de teste oficiais. Para isolar o efeito visual, conduzimos experimentos *zero-shot* [15] em 181 questões, comparando enunciados textuais puros com versões contendo imagens (Base64) de cadernos da OBI entre 1999 e 2025.

2 Trabalhos Relacionados

Estudos recentes demonstraram a capacidade de Grandes Modelos de Linguagem na geração de código [3, 18] e na resolução de exames educacionais [9, 14]. Com o avanço das técnicas de *Visual Instruction Tuning* [7], trabalhos têm avaliado Modelos de Linguagem Multimodais em raciocínio matemático com imagens [8] e tarefas complexas de Engenharia de Software [10, 16]. Destaca-se também a ideia de adicionar elementos visuais em enunciados de avaliações universitárias [1]. No entanto, apesar desses avanços, o impacto e a eficácia de MLLMs no raciocínio algorítmico de nível

educacional inicial, como os desafios da Olimpíada Brasileira de Informática (OBI), permanecem pouco explorados, motivando o presente estudo a avaliar a combinação de enunciados textuais e representações visuais.

3 Metodologia

Esta seção descreve o ambiente automatizado desenvolvido para avaliar a capacidade das MLLMs na resolução de problemas da modalidade Programação da OBI. O objetivo da metodologia é garantir um processo sistemático, reproduzível e comparável para análise do desempenho dos modelos. Para reduzir variações no processo experimental, foi utilizado um formato padronizado de prompt, contendo:

- O enunciado do problema com tags das imagens;
- A especificação de entrada e saída;
- Exemplos do enunciado;
- Instruções para gerar uma solução completa em código.

3.1 Visão Geral da Arquitetura

O sistema foi projetado para ser simples e automatizado, de forma que, ao escolher entre as linguagens de programação (C++ ou Python) e técnicas de prompt (Zero-Shot [15] ou Few-Shot [2]), o sistema deve produzir um código que será averiguado por meio do gabarito. Assim, haverá um agente codificador (LLM Multimodal) e um avaliador (*Judge*) para a dinamização do *benchmarking* dos modelos utilizados para resolver os problemas da OBI de 1999 a 2025, disponíveis no site [12], conforme ilustrado na Figura 1.

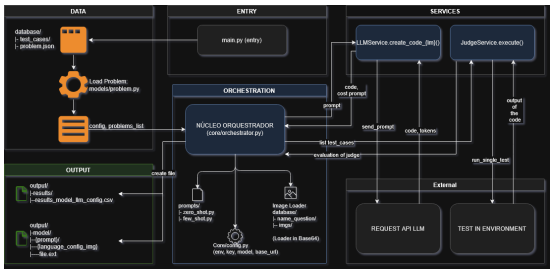

Figura 1: Arquitetura Experimental para Resolução de Problemas de Programação da OBI.

3.1.1 Banco de Problemas.

Para a realização do experimento, foi construído um banco de problemas (*dataset*) a partir do site oficial da Olimpíada Brasileira de Informática (OBI), utilizando *script* em Python e convertendo os cadernos de questões na modalidade de programação em arquivos JSON por meio do modelo Gemini 3.1 Pro. As questões extraídas, as provas, os gabaritos e a metodologia de extração estão disponíveis no repositório do GEMA-LAB [5] no formato zip ou executando o script python.

3.1.2 Camada Service.

Para o funcionamento do sistema, foi necessário criar dois *services* que se comunicam com o ambiente externo da arquitetura. O *service* do LLM tem como objetivo realizar requisições à API das LLMs, calculando o custo total do prompt, além do total de tokens gastos e o código gerado pelas LLMs. Entretanto, o *service* do *Judge* tem como objetivo avaliar o código por meio de *subthreads*, executando os casos de teste diretamente no ambiente de processamento. Com isso, o ambiente deve conter o g++ e algum interpretador Python para garantir a automatização do processo. Além disso, o *Judge* informa o status final da questão, a quantidade de acertos e de outros tipos de erros, e o tempo máximo de execução entre todos os casos de teste.

3.1.3 Orquestrador e Configuração.

Para garantir a execução contínua, criou-se um orquestrador para executar no momento em que o usuário escolher uma configuração para execução da automatização. Esse orquestrador tem como objetivo executar as seguintes tarefas: criação e edição de arquivos; carregamento dos casos de teste da questão; validação do código; carregamento das imagens; e iteração com as questões da OBI para cada LLM. Além disso, é importante a configuração de variáveis de ambiente para a conexão com as LLMs. Para garantir o funcionamento, é necessário informar: `NOME__API_KEY` (chave da API), `NOME__BASE_URL` (URL de comunicação), `NOME__MODEL_NAME` (nome do modelo) e o custo por token (`NOME__INPUT_PRICE` e `NOME__OUTPUT_PRICE`).

3.2 Modelos Avaliados

Os experimentos utilizaram cinco modelos proprietários capazes de gerar código a partir de descrições textuais: Claude Sonnet 4.6 (Anthropic), Gemini 3.1 Pro (Google DeepMind), GPT 5.4 (OpenAI), Grok 4.20 (X) e Mistral Small 4.

3.3 Ambiente de Avaliação

O processo de avaliação das soluções geradas pelos modelos tem as seguintes etapas: seleção de um problema do banco de dados; envio do enunciado ao modelo de linguagem; recebimento do código gerado pelo modelo; compilação e execução do código; avaliação automática utilizando casos de teste.

A verificação das soluções é realizada comparando a saída produzida com o gabarito dos casos de teste. Cada solução recebe um veredito comum em juízes automáticos: *Accepted* (AC, solução correta), *Wrong Answer* (WA, saída incorreta), *Compilation Error* (CE, erro de compilação), *Runtime Error* (RE, erro de execução) ou *Time Limit Exceeded* (TLE, excedeu o limite de tempo).

O limite de tempo para execução foi padronizado em 3 segundos para todas as instâncias e modelos deste estudo, pois essa informação não estava disponível nos cadernos de problemas da OBI.

3.4 Configuração do Experimento

No intuito de avaliar a capacidade das LLMs multimodais, foi necessário separar 181 questões que possuem imagens e casos de teste na base de problemas, sendo que essas questões compreendem 67 fáceis, 81 médias e 33 difíceis, segundo o Gemini 3.1 Pro.

Na configuração utilizada, o código deve ser produzido na linguagem Python com a técnica de *prompt zero-shot* [15]. Dessa forma, a imagem é inserida no prompt no formato Base64, com referências ao longo do enunciado, como ilustrado na Figura 2.

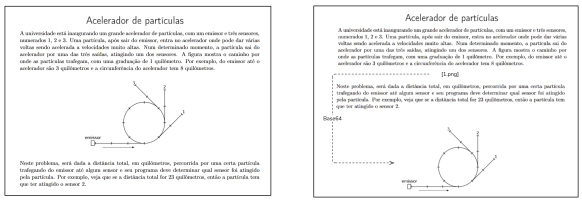

Figura 2: Formatação do Enunciado com Imagens em Base64.

Com isso, seguindo o pipeline ilustrado na Figura 3, para cada questão, o enunciado (com imagens em Base64, conforme Figura 2) é inserido no template do prompt. O agente gera um arquivo em Python, e o *Judge* valida os casos de teste salvando os resultados em um arquivo CSV.

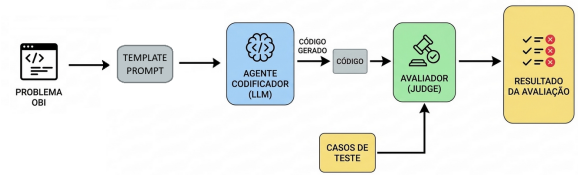

Figura 3: Pipeline de Execução para cada Questão.

Por fim, o ambiente deve ter acesso à internet para requisições, juntamente com um interpretador Python. O ambiente de *hardware* é constituído por um processador Intel Core i7-1360P de 13ª geração (2,20 GHz), 16 GB de memória RAM (4800 MT/s) e acelerador gráfico integrado Intel Iris Xe, operando sob o sistema Windows 11 Home (64 bits). Assim, o experimento teve a seguinte sequÊncia: primeiramente foi executado os enunciados sem as imagens e posteriormente com as imagens inseridas no prompt como mostra a Figura 2 para fins de comparação.

4 Resultados

Esta seção tem como objetivo apresentar os resultados do experimento executado. Nas Subseções 4.1 e 4.2 serão mostradas as estatísticas gerais dos LLMs multimodais. Já a Subseção 4.3 detalha os resultados para os casos de teste em tópicos avançados, como grafos e programação dinâmica.

4.1 Resultados dos Enunciados Sem Imagens

O desempenho em relação à taxa de acerto para questões de nível fácil (acima de 77%) e de nível médio (acima de 67%) é considerado alto, como visto na Tabela 1, destacando-se os modelos Claude Sonnet 4.6, GPT 5.4 e Gemini 3.1 Pro. Entretanto, apenas o modelo Gemini 3.1 Pro obteve um desempenho satisfatório para questões consideradas difíceis. Vale ressaltar que este modelo também exigiu o maior tempo de processamento para a geração da resolução e o maior custo médio. Com isso, uma característica observada durante o experimento é que os modelos ainda têm dificuldades cognitivas para a resolução de problemas complexos. Um simples *prompt* em configuração *zero-shot* pedindo para resolver a questão não foi suficiente para passar no gabarito, pois uma característica intrínseca da programação competitiva é a necessidade de prever casos extremos (*corner cases*) de entrada e saída.

Tabela 1: Comparativo de Desempenho dos Modelos Multimodais sem Imagens

Modelo	Fácil AC (%)	Médio AC (%)	Difícil AC (%)	Tokens I/O (Média)	Custo (US$)	Tempo (s)
Claude Sonnet 4.6	77.61	74.07	36.36	1698	0.0121	9
GPT 5.4	80.60	67.90	27.27	1123	0.0059	4
Gemini 3.1 Pro	97.01	87.65	75.76	5128	0.0520	38
Grok 4.20	61.19	53.09	18.18	1958	0.0078	9
Mistral Small 4	58.21	48.15	9.09	1212	0.0016	3

4.2 Resultados dos Enunciados Com Imagens

Ao aumentar o nível de complexidade do *prompt* com a adição de imagens, apenas o modelo Gemini 3.1 Pro manteve os mesmos percentuais da Tabela 1. No entanto, ao analisar os demais modelos multimodais, observou-se um aumento na taxa de acerto para questões de nível fácil. Entretanto, para questões de nível médio, os modelos Gemini 3.1 Pro e Grok 4.20 mantiveram o mesmo desempenho, o GPT 5.4 obteve leve aumento, e os demais apresentaram queda na taxa de acerto. Além disso, para questões de nível difícil, apenas os modelos Gemini 3.1 Pro, GPT 5.4 e Mistral Small 4 conseguiram acertar mais questões utilizando o contexto visual. Isso evidencia que o aumento da complexidade do *prompt* não é garantia de resolução para questões de níveis médio e difícil, como demonstrado na Tabela 2, embora possa servir como um guia para a modelagem da solução.

Tabela 2: Comparativo de Desempenho dos Modelos Multimodais com Imagens

Modelo	Fácil AC (%)	Médio AC (%)	Difícil AC (%)	Tokens I/O (Média)	Custo (US$)	Tempo (s)
Claude Sonnet 4.6	83.58	71.60	30.30	2049	0.0148	12
GPT 5.4	82.09	69.14	30.30	1329	0.0065	4
Gemini 3.1 Pro	97.01	87.65	78.79	6539	0.0567	39
Grok 4.20	73.13	53.09	15.15	1792	0.0061	7
Mistral Small 4	61.19	41.98	12.12	1456	0.0018	3

4.3 Desempenho em Tópicos Avançados

Cabe ressaltar que, para o *Judge* aceitar uma submissão, o código deve passar em todos os casos de teste da questão. Ao analisar especificamente as questões que envolvem tópicos de grafos e programação dinâmica (64 questões e 2.491 casos de teste no total), é perceptível na Figura 4 que a utilização de imagens no enunciado proporcionou uma melhora no desempenho de acerto individual dos casos de teste. As exceções foram os modelos Claude Sonnet 4.6 e Mistral Small 4, que apresentaram um desempenho inferior ao processar o contexto visual.

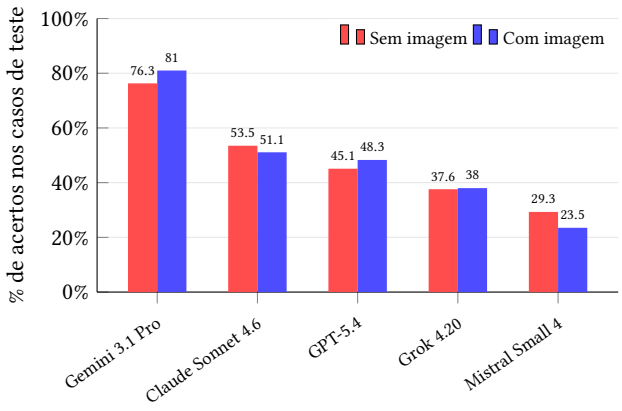

Figura 4: Desempenho comparativo de acertos totais por modelo com e sem imagem.

5 Conclusão e Trabalhos Futuros

Este trabalho apresentou uma arquitetura automatizada e um banco inédito de 493 questões para *benchmarking* de MLLMs na OBI. Os experimentos com 181 problemas multimodais (*zero-shot*, Python) indicam que a inclusão de imagens auxilia na compreensão semântica beneficiando questões de nível fácil e tópicos avançados na maioria dos modelos, mas não é suficiente para superar limitações em raciocínios algorítmicos complexos. A tensão entre o formato gerado pelos modelos probabilísticos e o determinismo do *Judge* permanece como um desafio relevante na geração autônoma de código.

Como trabalhos futuros, sugerimos a replicação em *datasets* diversificados para validar a capacidade de generalização e a experimentação com outras linguagens (como C++). A adoção de técnicas avançadas de *prompting* (ex: *Few-Shot* e *Chain-of-Thought*) aliada a fluxos iterativos de *feedback* com o *Judge* são caminhos promissores para melhorar as taxas de acerto e aproximar a avaliação ao fluxo real de resolução de problemas de programação e Engenharia de Software.

Disponibilidade de Artefatos

A arquitetura experimental, juntamente com os códigos e dados utilizados nas avaliações, encontra-se disponível para fins de revisão em: https://github.com/GEMA-LAB/obi-benchmarking

Referências

[1] Thales Sales Almeida, Thiago Laitz, Giovana Kerche Bonás, and Rodrigo Nogueira. 2023. BLUEX: A benchmark based on Brazilian Leading Universities Entrance eXams. *arXiv preprint arXiv:2307.05410* abs/2307.05410 (2023). https://api.semanticscholar.org/CorpusID:259765915

[2] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, Vol. 33. 1877–1901.

[3] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Pond'e de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mo Bavarian, Clemens Winter, Phil Tillet, Felipe Petroski Such, David W. Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William H. Guss, Alex Nichol, Igor Babuschkin, Suchir Balaji, Shantanu Jain, Andrew Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew M. Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. 2021. Evaluating Large Language Models Trained on Code. *arXiv.org* abs/2107.03374 (July 2021), 29 pages. doi:10.48550/arXiv.2107.03374

[4] Chaoyou Fu, Peixian Chen, Yunhang Shen, Yulei Qin, Mengdan Zhang, Xu Lin, Zhenyu Qiu, Wei Lin, Jinrui Yang, Xiawu Zheng, Ke Li, Xing Sun, and Rongrong Ji. 2023. MME: A Comprehensive Evaluation Benchmark for Multimodal Large Language Models. *arXiv preprint arXiv:2306.13394* abs/2306.13394 (2023). https://api.semanticscholar.org/CorpusID:259243928

[5] GEMA-LAB. 2026. Problems of the OBI Dataset. https://github.com/GEMA-LAB/problems-of-the-obi.git Repositório GitHub.

[6] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. 2020. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361* (2020). https://arxiv.org/abs/2001.08361

[7] Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. 2023. Visual Instruction Tuning. *ArXiv* abs/2304.08485 (2023). https://api.semanticscholar.org/CorpusID:258179774

[8] Pan Lu, Hritik Bansal, Tony Xia, Jiacheng Liu, Chun yue Li, Hannaneh Hajishirzi, Hao Cheng, Kai-Wei Chang, Michel Galley, and Jianfeng Gao. 2023. MathVista: Evaluating Mathematical Reasoning of Foundation Models in Visual Contexts. In *International Conference on Learning Representations (ICLR)*. https://api.semanticscholar.org/CorpusID:264491155

[9] Nabor C. Mendonça. 2024. Evaluating ChatGPT-4 Vision on Brazil's National Undergraduate Computer Science Exam. *ACM Transactions on Computing Education* 24 (2024), 1 – 56. https://api.semanticscholar.org/CorpusID:270521438

[10] Isela Mendoza, Fernando Silva Filho, Gustavo Medeiros, Aline Paes, and Vânia O. Neves. 2024. Comparative Analysis of Large Language Model Tools for Automated Test Data Generation from BDD. In *Anais do XXXVIII Simpósio Brasileiro de Engenharia de Software (SBES)*. Sociedade Brasileira de Computação - SBC, Porto Alegre, RS, Brasil, 280–290. doi:10.5753/sbes.2024.3423

[11] Desnes Nunes, Ricardo Primi, Ramon Pires, Roberto Lotufo, and Rodrigo Nogueira. 2023. Evaluating GPT-3.5 and GPT-4 models on Brazilian university admission exams. *arXiv preprint arXiv:2303.17003* (2023).

[12] Olimpíada Brasileira de Informática. 2025. Provas e Gabaritos de Edições Passadas. https://olimpiada.ic.unicamp.br/passadas/ Acessado em: 27 abr. 2026.

[13] Esdras Caleb Oliveira Silva, Roberta de Souza Coelho, and Lyrene Fernandes da Silva. 2025. LLMs as Test Generators: A Comparative Benchmarking Study. In *Anais do XXXIX Simpósio Brasileiro de Engenharia de Software (SBES)*. Sociedade Brasileira de Computação - SBC, Porto Alegre, RS, Brasil, 25–36. doi:10.5753/sbes.2025.9618

[14] Cayo Viegas, Rohit Gheyi, and Márcio Ribeiro. 2025. Assessing the Capability of LLMs in Solving POSCOMP Questions. *arXiv preprint arXiv:2505.20338* abs/2505.20338 (2025). https://api.semanticscholar.org/CorpusID:278911577

[15] Jason Wei, Maarten Bosma, Vincent Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M Dai, and Quoc V Le. 2022. Finetuned language models are zero-shot learners. In *International Conference on Learning Representations*. https://openreview.net/forum?id=gEZrGCozdqR

[16] John Yang, Carlos E. Jimenez, Alex L. Zhang, Kilian Adriano Lieret, Joyce Yang, Xindi Wu, Ori Press, Niklas Muennighoff, Gabriel Synnaeve, Karthik R. Narasimhan, Diyi Yang, Sida Wang, and Ofir Press. 2024. SWE-bench Multimodal: Do AI Systems Generalize to Visual Software Domains? *arXiv preprint arXiv:2410.03859* abs/2410.03859 (2024). https://api.semanticscholar.org/CorpusID:273185542

[17] Xiang Yue, Yuansheng Ni, Kai Zhang, Tianyu Zheng, Ruoqi Liu, Ge Zhang, Samuel Stevens, Dongfu Jiang, Weiming Ren, Yuxuan Sun, Cong Wei, Botao Yu, Ruibin Yuan, Renliang Sun, Ming Yin, Boyuan Zheng, Zhenzhu Yang, Yibo Liu, Wenhao Huang, Huan Sun, Yu Su, and Wenhu Chen. 2023. MMMU: A Massive Multi-Discipline Multimodal Understanding and Reasoning Benchmark for Expert AGI. *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2023), 9556–9567. https://api.semanticscholar.org/CorpusID:265466525

[18] Zihan Zheng, Zerui Cheng, Zeyu Shen, Shang Zhou, Kaiyuan Liu, Hansen He, Dongruixuan Li, Stanley Wei, Hang Hao, Jianzhu Yao, Peiyao Sheng, Zixuan Wang, Wenhao Chai, Aleksandra Korolova, Peter Henderson, Sanjeev Arora, Pramod Viswanath, Jingbo Shang, and Saining Xie. 2025. LiveCodeBench Pro: How Do Olympiad Medalists Judge LLMs in Competitive Programming? *arXiv preprint arXiv:2506.11928* abs/2506.11928 (2025). https://api.semanticscholar.org/CorpusID:279392173