AutoAC: Agentic Self-Refinement for Competitive Programming

Sriram Sai Ganesh
The Ohio State University
saiganesh.3@osu.edu

Robert Pham
The Ohio State University
pham.495@osu.edu

Abstract

We introduce **AutoAC**, an agent-computer interface (ACI) for agentic self-refinement in competitive programming that couples LLM code generation with grounded feedback from an online-judgestyle execution loop. AutoAC extends a baseline "prompt-solution-judge" workflow with (i) multi-agent chain-of-thought specialization and (ii) automated stress testing. Experiments on 300 problems from the CSES Problem Set across multiple open LLM families show that domain-specific CoT prompting improves partial correctness and that stress-testing yields substantial gains for stronger models; for example, on Qwen2.5-14B-Instruct, adding stress-testing to the best prompt condition increases performance by **+5.97% NTPR** and **+6.33% SR** (absolute). Overall, **AutoAC** demonstrates that embedding competitive-programming debugging practices into an ACI enables measurable improvements in reliable code generation under hidden-test evaluation.

1 Introduction

Large Language Models (LLMs) have established state-of-the-art performance on a variety of Natural Language Processing (NLP) tasks. They are increasingly adopted in digital environments, even for multifaceted problem-solving tasks like Software Engineering (SWE). Given this popular use case, it is natural to derive utility from examining frameworks and benchmarks that enhance LLMs' ability to code. Competitive Programming (CP), while providing a benchmark for reasoning that is easy to test automatically and a good proxy for real-world programming and reasoning [7], has also been found to be present in abundance in training data.

Hence, we find utility in formalizing an agentic feedback loop grounded in CP strategy that enables generalization of LLM CP beyond compositions of algorithms and data structures seen in popular datasets.

As an LLM evaluation metric, the advantages of CP are twofold:

1. Compositionality: It is known that LLMs struggle with tasks requiring a composition of complex skills [19]. Most algorithmic CP problems at a competitive level require the application of multiple algorithms and data structures in a specific order to achieve the necessary program correctness and efficiency. Hence, CP has emerged as a testing ground for this skill composition at the popular code-generation task.

2. Automated Evaluation: A programmer submitting solutions in a CP contest has their code tested by an automated Online Judge (OJ) system which compiles and executes their code against hidden test cases, returning a "Compilation Error" (CE), "Wrong Answer" (WA), "Runtime Error" (RE), "Time Limit Exceeded" (TLE), or "Accepted" (AC) result. This automated judgment of code correctness and efficiency enables a simple and automated testing loop, unlike broader SWE benchmarks requiring bespoke test cases for each type of problem [8].

However, preliminary experiments have revealed that popular CP datasets are represented in training data for LLMs, resulting in test leakage and an inaccurate picture of coding performance from benchmarks. This observation, along with the utility of CP as an evaluation metric, motivates our work.

2 Related Work

2.1 LLMs for Code Generation

LLMs have been studied for Software Engineering [5, 20, 8, 3] for their potential to automate tasks common in the maintenance of large codebases. In contrast to many other applications of LLMs, software engineers have access to automat-

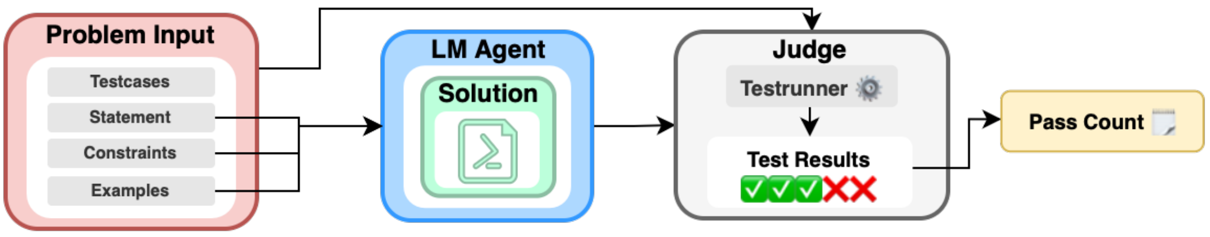

Figure 1: Visualization of the baseline ACI workflow that we use as a point of comparison.

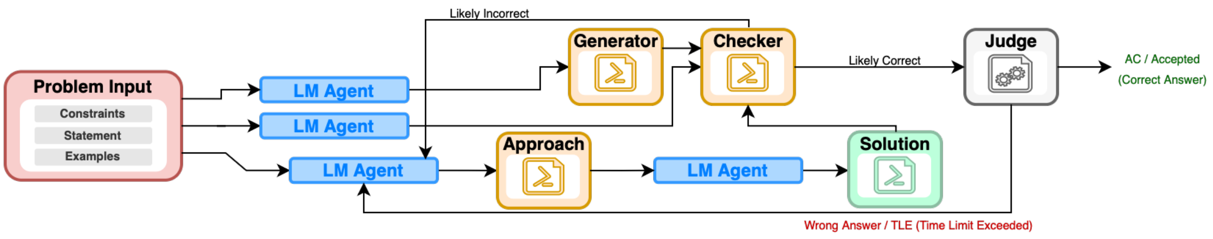

Figure 2: Extended ACI workflow with Stress-Testing behavior.

able ground truth (software execution) to evaluate software engineering artifacts. Moreover, the software engineering research community has already developed automated and semi-automated techniques for checking potentially incorrect results [3].

However, while these techniques test specific practical applications of LLMs like on benchmarks drawn from GitHub issues [8], they may not capture an accurate picture of the reasoning ability of these models.

2.2 Agent Computer Interfaces (ACIs)

Coupling LLMs with a grounded feedback signal in the form of an Agent Computer Interface (ACI) serves to significantly enhance their productivity [20, 6]. Specifically, ACIs tailored specifically for LMs outperform existing user interfaces (UIs) designed for human users [20] – a fact we leverage with our framework that uses guardrails and specific simple actions that the LLM can perform in order to achieve optimal performance.

3 Methodology

3.1 Overview

We present **AutoAC**, an agent-computer interface (ACI) that allows an LLM agent to interact, dissect, plan and solve a given competitive programming problem. **AutoAC** will initially feed problem information into the LLM, receive a prompt, and iterate upon the prompt. As a baseline, the ACI will simply and concisely prompt the LLM for a solution to the problem. Against this baseline, we propose a more comprehensive inter-agent communication framework, test generator, and a solution verifier, to iterate & improve on failed solutions. Further extensions may include finetuning the LLM to improve its performance specifically on competitive programming problems, as well as implementing chain-of-thought (CoT) reasoning into the ACI, which tends to improve the performance of LLMs on complex reasoning tasks [18].

See Figure 1 for the workflow of the overall baseline agent-computer interface (ACI). Statement, constraints, and example correspond to θ, λ, and δ_S respectively as defined in 4.1. Test cases correspond to δ_H. All problem input information is fed into the language model (LM) agent to be prompted for a solution. The judge, using the given test case information, verifies the LM agent's solution against all given test cases and outputs the verdicts of the solution.

3.2 Chain-of-Thought Reasoning

See Figure 2 for a visualization of the components that will replace part of the baseline ACI workflow. Initially, chain-of-thought (CoT) reasoning was implemented to improve the LM agent's problem-solving ability. Here, one LM agent instance is tasked with designing and planning out approaches to solve the problem optimally. Another LM agent instance is tasked with taking an approach and implementing the final solution in code, requiring more attention to finer details.

3.3 Automated Stress Testing

Within Competitive Programming, stress-testing is a technique used to debug a solution with a minimal test case causing failure. This allows the programmer to quickly discover flaws in the correctness of the approach and/or solution implementation. Due to some test cases being hidden from the programmer (see 4.1), stress-testing becomes invaluable for programmers debugging solutions on their own. In live competitions, many programmers create tests by hand, however, automating this process by quickly writing another stress-test-generating program can also be an effective strategy. We use an ACI for this purpose. Given the problem constraints, problem statement, and test examples, LM agent instances program a test case generator and an output verifier. All additional synthetic test cases will be small enough to be computationally feasible to verify with a naive brute-force comparison written by the LM agent.

4 Formal Definitions

4.1 Problem Statement

A competitive programming problem $\mathcal{P}$ is parametrized by a problem statement θ, input constraints λ, a sample test case set δ_S and a hidden test case set δ_H. Each test case set consists of program input-output pairs

$$\delta \in \{(\epsilon^{(i)}, \nu^{(i)}) \mid i \in [1, 2, \ldots, k], k \geq 1\}$$

A dataset consists of N such programming problems, Dataset $\mathcal{D} : \{\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_N\}$

4.2 Solution

An Accepted (AC) solution $\mathcal{S}$ to a competitive programming problem $\mathcal{P} : \{\theta, \lambda, \delta_S, \delta_H\}$ is a function $\mathcal{S}$ such that

$$\nu \equiv \mathcal{S}(\epsilon) \ \forall (\epsilon, \nu) \ \in \ (\delta_S \cup \delta_H)$$

That is, for all test case input-output pairs (ϵ, ν) in $\delta_S \ \cup \ \delta_H$, we expect that $\nu \equiv \mathcal{S}(\epsilon)$. Note that we use the symbol for equivalence ($\equiv$) as opposed to equality, since two given correct floating-point answers may not exactly match due to computational errors.

4.3 Candidate Solutions

Given an instance of a programming problem $\mathcal{P} : \{\theta, \lambda, \delta_S, \delta_H\}$, a programmer is given a partial view $\mathcal{Q} : \{\theta, \lambda, \delta_S\}$, with the goal of writing a candidate solution $\mathcal{S}_C$ that satisfies the Solution criteria above.

4.4 Evaluation Metrics

Considering a dataset of competitive programming problems $\mathcal{D} : \{\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_N\}$ we evaluate the performance of a programming agent on a several metrics:

Solve Rate (SR): Programmers are primarily evaluated on Solve Rate – the proportion of solved problems. A problem is considered solved if, for *every* test case, running the candidate solution results in an answer matching the expected result. These solutions are considered Accepted (AC).

$$\text{AC}_i \leftarrow \nu \equiv \mathcal{S}_C^{(i)}(\epsilon) \ \forall (\epsilon, \nu) \ \in \ \left(\delta_S^{(i)} \cup \delta_H^{(i)}\right)$$

$$\text{N}_{\text{AC}} \leftarrow \sum_{i=1}^{|D|} \mathbb{1}(\text{AC}_i)$$

$$\text{SR} \leftarrow 100 \cdot \frac{\text{N}_{\text{AC}}}{|D|}$$

$\mathbb{1} \rightarrow$ Indicator function

That is, we measure the proportion of problem solutions that pass *all* of the corresponding problem's test cases & receive an Accepted (AC) judgement.

Normalized Test-case Pass Rate (NTPR): Many of the CP problems in consideration for this project are rated higher than the CodeForces Elo of any current LLM [13]. Hence, to gain some notion of solve rates even when many solutions are judged Wrong Answer (WA), we compute partial pass rate as follows:

$$K \leftarrow |\delta_S \cup \delta_H|$$

$$\text{Partial}^{(i)} \leftarrow \frac{\sum_{j=1}^{K} \mathbb{1}\left(\nu^{(j)} \equiv \mathcal{S}^{(i)}(\epsilon^{(j)})\right)}{K}$$

$$\text{NTPR} \leftarrow 100 \cdot \frac{\sum_{i=1}^{|D|} \text{Partial}^{(i)}}{|D|}$$

That is, we compute the mean of the proportion of test case passes per-problem in $\mathcal{D}$.

Experiment		**Detailed Prompt Description**
Tag	Label	
A	Baseline non-CoT	Instruct the model to solve the problem in C++
B	Baseline CoT	Instruct the model to reason through its solution.
C	Refined CoT	Step-by-step CoT Instructions
D	Domain-specific CoT	Detailed CoT including guidelines about CP best practices
E	ICL 1-shot OOD Short	ICL + CoT + 1 concise synthetic problem-solution pair
F	ICL 1-shot OOD Long	ICL + CoT + 1 complex synthetic problem-solution pair
G	ICL 1-shot ID Long	ICL + CoT + 1 in-domain (from training data) problem-solution pair
H	ICL 3-shot OOD Short	ICL + CoT + 3 examples of concise synthetic problem-solution pairs
A* to **H** *	Stress-test variants	Any of experimental groups above + iteration & testcase generation

Table 1: Descriptions of the prompt setting for each experimental group. ICL refers to In-Context Learning. ID refers to In-Distribution examples. OOD refers to Out-of-Distribution examples.

Large Language Model	**Normalized Test-case Pass Rate (%)**							
	A	**B**	**C**	**D**	**E**	**F**	**G**	**H**
Qwen2.5-0.5B-Instruct	2.01	1.62	1.64	0.87	5.09	3.94	5.85	**6.86**
Llama-3.2-1B-Instruct	3.85	7.48	7.65	7.97	6.77	7.20	7.69	**9.68**
Qwen2.5-1.5B-Instruct	4.48	5.61	4.08	6.37	9.49	5.47	**10.86**	10.05
Llama-3.2-3B-Instruct	9.98	11.35	13.52	13.32	13.23	13.36	16.39	**18.21**
Qwen2.5-3B-Instruct	14.70	12.88	12.65	17.15	17.56	14.32	16.93	**17.77**
Qwen2.5-7B-Instruct	23.11	25.04	24.56	25.85	**28.97**	23.85	26.49	28.29
CodeLlama-7b-Instruct-hf	12.05	12.92	6.90	7.82	12.67	7.64	10.72	**12.94**
Llama-3.1-8B-Instruct	20.19	17.83	16.78	17.26	20.12	14.82	19.64	**20.67**
CodeLlama-13b-Instruct-hf	15.55	16.64	14.87	11.31	12.98	10.55	**17.68**	17.36
Qwen2.5-14B-Instruct	28.80	26.06	28.48	**34.50**	29.30	29.57	28.81	32.60

Table 2: % Normalized Test-case Pass Rate (NTPR) for each model across experiments. We highlight results from the best-performing experiment for each language model.

Large Language Model	**Solve Rate (%)**							
	A	**B**	**C**	**D**	**E**	**F**	**G**	**H**
Qwen2.5-0.5B-Instruct	0.33	0.00	0.33	0.00	0.33	0.33	**1.00**	0.33
Llama-3.2-1B-Instruct	0.00	0.66	0.66	**1.00**	0.00	**1.00**	**1.00**	0.33
Qwen2.5-1.5B-Instruct	0.66	1.33	1.00	1.00	1.33	0.66	1.33	**1.66**
Llama-3.2-3B-Instruct	2.66	2.66	3.66	3.33	2.66	1.66	3.66	**4.33**
Qwen2.5-3B-Instruct	6.00	3.66	3.00	**7.33**	5.66	4.66	5.00	4.66
Qwen2.5-7B-Instruct	9.66	10.66	10.00	12.00	**13.66**	9.33	12.33	12.66
CodeLlama-7b-Instruct-hf	0.66	1.00	0.33	**1.66**	1.33	0.66	1.00	1.00
Llama-3.1-8B-Instruct	**7.00**	4.66	4.66	6.00	6.00	3.33	5.33	5.66
CodeLlama-13b-Instruct-hf	1.33	2.00	2.00	1.00	1.33	1.00	**3.00**	2.33
Qwen2.5-14B-Instruct	15.66	13.00	13.33	**18.66**	14.33	14.33	16.33	16.33

Table 3: % Solve Rate (SR) for each model across experiments. We highlight results from the best-performing experiment for each language model.

Large Language Model	Stress Test Normalized Testcase Pass Rate (%)							
	D	D*	E	E*	G	G*	H	H*
Qwen2.5-3B-Instruct	17.15	13.33	17.56	**22.59**	16.93	20.35	17.77	20.98
Qwen2.5-14B-Instruct	34.50	**40.47**	29.30	32.76	28.81	38.39	32.60	33.43
	Stress Test Solve Rate (%)							
Qwen2.5-3B-Instruct	7.33	5.00	5.66	**12.00**	5.00	8.00	4.66	8.00
Qwen2.5-14B-Instruct	18.66	**25.00**	14.33	20.00	16.33	23.00	16.33	19.00

Table 4: % Stress-testing results for each model across experiments. The asterisk * denotes the addition of stress-testing to the experimental group.

Temperature	NTPR	SR
0.085	**0.362**	**0.180**
0.170	0.350	0.176
0.255	0.308	0.160
0.340	0.349	**0.180**
0.425	0.327	0.163
0.510	0.341	0.176
0.595	0.318	0.160
0.680	0.308	0.167
0.765	0.314	0.170

Table 5: % Experimental results for experiment **D** – Domain-specific CoT, for the Qwen-2.5-14B-Instruct model, across sampling temperatures.

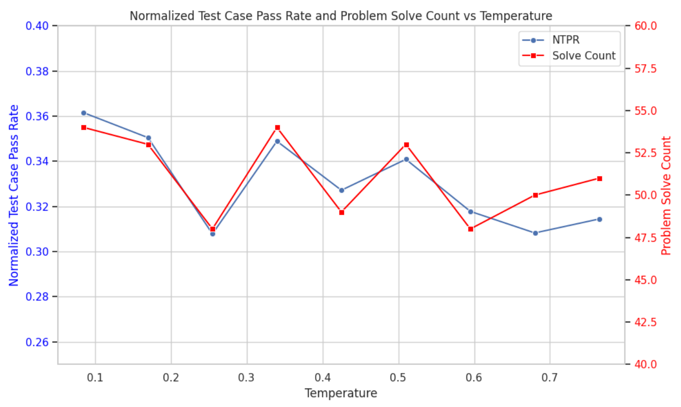

Figure 3: Ablating the effect of temperature on the performance of **AutoAC**. We observe a weak negative correlation between temperature and SR & NTPR.

Language Model	Average # of Reprompt Attempts	NTPR	SR	Number of non-first-attempt solves
Qwen2.5-0.5B-Instruct	2.97	0.089	0.01	0
Qwen2.5-1.5B-Instruct	2.91	0.118	0.03	0
Qwen2.5-3B-Instruct	2.91	0.151	0.03	0
Qwen2.5-7B-Instruct	2.49	0.310	0.17	0
Qwen2.5-14B-Instruct	2.58	0.302	0.14	0

Table 6: % Experimental results for experiment **H** – 3-shot synthetic code sample CoT, for the Qwen family of models, with a max of 4 attempts per question (3 retries).

5 Experimental Setup

5.1 Models

The nature of our framework enables each ACI to support any model hosted locally or via an API call. Here, we answer a series of braod experimental questions with a variety of model families.

For the first set of results, we consider Meta's Llama (3.1-8B-Instruct, 3.2-1B-Instruct, 3.2-3B-Instruct) [4], CodeLlama (13b-Instruct-hf, 7b-Instruct-hf), and Qwen2.5 (0.5B-Instruct, 1.5B-Instruct, 3B-Instruct, 7B-Instruct, 14B-Instruct) [14].

We proceed to explore experimental choices for our implementation of **AutoAC**. We consider the effect of sampling temperature, reprompting, stress-testing (ie. iterating on solutions), and an overall 'most-improved' experimental group. To reduce the computational costs of running these experiments, we constrain our problem space to a 33% split of our data, and only run experiments on the `Qwen-2.5-*B-Instruct` [14] family of models.

5.2 Dataset

Our dataset consists of 300 algorithmic programming practice problems from the CSES Problem Set (`https://cses.fi/problemset/`). These problems range in difficulty from beginner-friendly to extremely challenging, with some solved by less than 0.07% of users.

5.3 Test Bed

All experiments are conducted on Ohio Supercomputing Center (OSC) clusters in full precision using the Floating Point 32 (`fp32`) data type. Models smaller than 7 billion parameters are inferenced on OSC's Pitzer cluster using NVIDIA Volta V100 GPUs with 32GB of VRAM. Larger models are inferenced on the Cardinal cluster using NVIDIA H100 GPUs with 96GB of VRAM.

5.4 Experimental Groups

We divide our experiments into a series of experimental groups, each with their own prompt. These groups are described in Table 1.

6 Results

Our current framework implements the comprehensive ACI workflow described in Figure 2. We make a series of observations, at two different scales, in trends of Solve Rate (SR) and Normalized Test-case Pass Rate (NTPR):

6.1 Broad Trends

Chain-of-Thought prompting has been known to elicit more correct responses in autoregressive language models [18]. We find unilaterally that prompting the model with Chain-of-Thought offers significant gains in NTPR; models that reason through their solutions tend to more often pass test cases.
The first half of our experiment groups test the efficacy of different CoT prompt conditions. Specifically, we compare a non-CoT baseline against a human-defined CoT prompt, one refined by a language model, and a third annotated with CP-specific guidelines. We find this final prompt – one that defines a thread of reasoning & code choices tailored to solving CP problems – outperforms all 3 other CoT settings.

In-Context-Learning has been known to improve the generalization of language models to new domains [2]. While all of these models have been pretrained on programming examples, CP problems tend to frame algorithmic problems in contexts that may deceive a naive agent as to the true nature of a problem, or bias it towards an incorrect approach. We find utility in providing a set of CP problem-solution pairs in-context. Further, in-context examples of reasoning and solution formation can assist with the model correctly structuring its output.
The latter half of the problem groups all implement ICL to different degrees, varying the length and number of examples, as well as the distribution from which they are drawn. We find that a variety of short examples (problems of lower complexity) offer promising performance gains compared to a single example. Further, we find no significant difference comparing ICL examples drawn from the training data and ICL examples generated by a different language model. In the future, we plan to conduct experiments varying the number of low-complexity synthetic examples to finalize a k number of examples in our final k-shot pipeline.

Model Size correlates strongly with performance, as existing literature has studied [9]. Our experiments corroborate this finding – we achieve the highest NTPR & SR performance in each of the 8 experiment groups with the

`Qwen2.5-14B-Instruct` language model, sometimes improving the SoTA among the rest by over 80%. Notably, an exception to this expected performance scaling is the `CodeLlama` family of models. These models are finetuned checkpoints of Llama 2 [17], a significantly less performant model from 2023 when compared to the other 2024/5 language models. We find that, despite being finetuned on coding tasks, CodeLlama models are outperformed by significantly smaller but more recent models – CodeLlama-13B-Instruct consistently achieves a lower SR than the 2025 checkpoint of Llama-3.2-3B, despite having 10B more parameters.

6.2 AutoAC Results

Temperature We hypothesized that modifying the model's temperature parameter would balance between determinism and exploration/creativity in the LLMs' results. Specifically, since we assume for there to be largely one correct approach to solving a problem, we initially operated under the assumption that increasing this parameter would result in a sharp decline in **AutoAC**'s NTPR and SR.

We experimentally ablate the effect of changing temperature. Specifically, we test an evenly spaced range of temperatures between 0.085 and 0.765, on the best-performing `Qwen-2.5-14B-Instruct` model. Results are in Table 5 and Figure 3.

Contrary to our earlier assumptions, we observe that performance stays fairly consistent, with a weak negative correlation between temperature and framework performance. In order to maximize the likelihood of obtaining high-quality results, we set temperature to $t = 0.1$ going forward.

Reprompting We consider the possibility that, given the non-determinism in model outputs that the sampling temperature parameter introduces, multiple attempts of the same problem generation can result in differing problem solutions. These additional solutions may pass testcases that the original approach does not.

Experimental results in table 6 demonstrate that reprompting provides no additional benefit to the NTPR & SR of our models. However, we note that our experiments were conducted with our previously set temperature of $t = 0.1$; it is possible that we can find gains in reprompting if we increase the temperature to improve the creativity of model generations.

Stress-testing In most cases, LLMs fail to solve a problem on their first attempt. Thus, we realized the need to create a systemic framework allowing models to self-correct and refine their original solutions as necessary. By implementing the workflow described in Figure 2, this agentic stress-testing behavior allowed models to solve more problems than the baseline described in Figure 1.

As seen in Table 4, stress-testing behavior resulted in significant improvements for most of the top-performing baseline prompts. Notably, `Qwen-2.5-14B-Instruct` consistently performs better in all metrics with stress-testing behavior, while `Qwen-2.5-3B-Instruct` performs better in all metrics three out of four times.

Largest Improvement Across all experiments, the consistently top-performing model `Qwen-2.5-14B-Instruct` achieves the highest overall NTPR and SR in experiment **D***. By setting temperature $t = 0.1$ and implementing stress-test behavior, we can see a $+5.97\%$ (abs) and $+6.33\%$ (abs) increase in NTPR and SR respectively.

7 Future Work

Larger Models may lead to improved performance. Current experiments are limited to models running locally on OSC, with our largest tested model containing 14B parameters. From Table 2, we can observe a general trend in greater parameter numbers resulting in greater pass rates. Thus, we hypothesize performance may be easily improved by leveraging larger models cloned locally or via API access to low-cost model providers such as Deepseek, among others.

Test-time scaling Existing work establishes that LLM finetuning on quality task-specific data results in much improved performance. Beyond naive finetuning, we consider potential benefits from a more recent test-time scaling paradigm.

The recent paradigm of scaling test-time compute [16] scaling shows promise for application in Competitive Programming. We are keen to explore the efficacy of Large Reasoning Models [12] as agents in the **AutoAC** framework, experiments we were not able to run due to budget constraints.

Group Relative Policy Optimization [15] presents a compelling new model preference alignment scheme that we see as being useful to finetune LLMs for competitive programming tasks, though it is possible that a scarcity of high-quality and diverse reasoning data for CP tasks is a bottleneck in this effort. The budget forcing scheme popularized by “Simple Scaling" (Snell et. al [10]) could present a means to mitigate the effect of this scarcity.

8 Acknowledgements

We are thankful for the opportunity and resources provided by Dr. Sachin Kumar during the SP’25 offering of `CSE 5525`.
We are also thankful to the Ohio Supercomputer Center The Ohio State University.

Appendix

A Implementation

This report understates the effort it takes to set up a codebase to enable this level of modularity, efficiency and scale across our broad range of experiments. As an example, Table 2 took 24,000 LLM-generated Competitive Programming solutions to curate.
All code is available at `github.com/Sriram-Sai-Ganesh/AutoAC` upon request. Please email the authors for access.

B Background: Competitive Programming

In Competitive Programming (CP), a problem typically consists of a narrative that sets the context, which models need to understand and convert into an algorithmic problem. [7] The challenge lies in comprehending the narrative, identifying the underlying algorithmic issues, and implementing an efficient solution in programming languages such as C++ and Java. [7] Accepted programs must satisfy stringent testing conditions, including producing outputs that exactly match with test cases, executing within memory limits, and terminating within time constraints. [7] Competition-level programming problems require advanced reasoning and mathematical modeling skills, essential for AI. [7]

Competitive programming problems are known to challenge reasoning abilities and can be complex, requiring advanced computational thinking and problem-solving [11]. Often, even if a solution is provably correct, it may not run within predefined time constraints [1], resulting in nontrivial algorithmic or mathematical optimizations being necessary to solve the problem. They can also be tested and verified automatically, making them good candidates to test the reasoning capability of LLMs [11], unlike pure math problems which require more intricate work to verify.

References

[1] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde 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, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter,

Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew 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. *Preprint*, arXiv:2107.03374.

[2] Qingxiu Dong, Lei Li, Damai Dai, Ce Zheng, Jingyuan Ma, Rui Li, Heming Xia, Jingjing Xu, Zhiyong Wu, Tianyu Liu, Baobao Chang, Xu Sun, Lei Li, and Zhifang Sui. 2024. A survey on in-context learning. *Preprint*, arXiv:2301.00234.

[3] Angela Fan, Beliz Gokkaya, Mark Harman, Mitya Lyubarskiy, Shubho Sengupta, Shin Yoo, and Jie M. Zhang. 2023. Large language models for software engineering: Survey and open problems. *Preprint*, arXiv:2310.03533.

[4] Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, Arun Rao, Aston Zhang, Aurelien Rodriguez, Austen Gregerson, Ava Spataru, Baptiste Roziere, Bethany Biron, Binh Tang, Bobbie Chern, Charlotte Caucheteux, Chaya Nayak, Chloe Bi, Chris Marra, Chris McConnell, Christian Keller, Christophe Touret, Chunyang Wu, Corinne Wong, Cristian Canton Ferrer, Cyrus Nikolaidis, Damien Allonsius, Daniel Song, Danielle Pintz, Danny Livshits, Danny Wyatt, David Esiobu, Dhruv Choudhary, Dhruv Mahajan, Diego Garcia-Olano, Diego Perino, Dieuwke Hupkes, Egor Lakomkin, Ehab AlBadawy, Elina Lobanova, Emily Dinan, Eric Michael Smith, Filip Radenovic, Francisco Guzmán, Frank Zhang, Gabriel Synnaeve, Gabrielle Lee, Georgia Lewis Anderson, Govind Thattai, Graeme Nail, Gregoire Mialon, Guan Pang, Guillem Cucurell, Hailey Nguyen, Hannah Korevaar, Hu Xu, Hugo Touvron, Iliyan Zarov, Imanol Arrieta Ibarra, Isabel Kloumann, Ishan Misra, Ivan Evtimov, Jack Zhang, Jade Copet, Jaewon Lee, Jan Geffert, Jana Vranes, Jason Park, Jay Mahadeokar, Jeet Shah, Jelmer van der Linde, Jennifer Billock, Jenny Hong, Jenya Lee, Jeremy Fu, Jianfeng Chi, Jianyu Huang, Jiawen Liu, Jie Wang, Jiecao Yu, Joanna Bitton, Joe Spisak, Jongsoo Park, Joseph Rocca, Joshua Johnstun, Joshua Saxe, Junteng Jia, Kalyan Vasuden Alwala, Karthik Prasad, Kartikeya Upasani, Kate Plawiak, Ke Li, Kenneth Heafield, Kevin Stone, Khalid El-Arini, Krithika Iyer, Kshitiz Malik, Kuenley Chiu, Kunal Bhalla, Kushal Lakhotia, Lauren Rantala-Yeary, Laurens van der Maaten, Lawrence Chen, Liang Tan, Liz Jenkins, Louis Martin, Lovish Madaan, Lubo Malo, Lukas Blecher, Lukas Landzaat, Luke de Oliveira, Madeline Muzzi, Mahesh Pasupuleti, Mannat Singh, Manohar Paluri, Marcin Kardas, Maria Tsimpoukelli, Mathew Oldham, Mathieu Rita, Maya Pavlova, Melanie Kambadur, Mike Lewis, Min Si, Mitesh Kumar Singh, Mona Hassan, Naman Goyal, Narjes Torabi, Nikolay Bashlykov, Nikolay Bogoychev, Niladri Chatterji, Ning Zhang, Olivier Duchenne, Onur Çelebi, Patrick Alrassy, Pengchuan Zhang, Pengwei Li, Petar Vasic, Peter Weng, Prajjwal Bhargava, Pratik Dubal, Praveen Krishnan, Punit Singh Koura, Puxin Xu, Qing He, Qingxiao Dong, Ragavan Srinivasan, Raj Ganapathy, Ramon Calderer, Ricardo Silveira Cabral, Robert Stojnic, Roberta Raileanu, Rohan Maheswari, Rohit Girdhar, Rohit Patel, Romain Sauvestre, Ronnie Polidoro, Roshan Sumbaly, Ross Taylor, Ruan Silva, Rui Hou, Rui Wang, Saghar Hosseini, Sahana Chennabasappa, Sanjay Singh, Sean Bell, Seohyun Sonia Kim, Sergey Edunov, Shaoliang Nie, Sharan Narang, Sharath Raparthy, Sheng Shen, Shengye Wan, Shruti Bhosale, Shun Zhang, Simon Vandenhende, Soumya Batra, Spencer Whitman, Sten Sootla, Stephane Collot, Suchin Gururangan, Sydney Borodinsky, Tamar Herman, Tara Fowler, Tarek Sheasha, Thomas Georgiou, Thomas Scialom, Tobias Speckbacher, Todor Mihaylov, Tong Xiao, Ujjwal Karn, Vedanuj Goswami, Vibhor Gupta, Vignesh Ramanathan, Viktor Kerkez, Vincent Gonguet, Virginie Do, Vish Vogeti, Vítor Albiero, Vladan Petrovic, Weiwei Chu, Wenhan Xiong, Wenyin Fu, Whitney Meers, Xavier Martinet, Xiaodong Wang, Xiaofang Wang, Xiaoqing Ellen Tan, Xide Xia, Xinfeng Xie, Xuchao Jia, Xuewei Wang, Yaelle Goldschlag, Yashesh Gaur, Yasmine Babaei, Yi Wen, Yiwen Song, Yuchen Zhang, Yue Li, Yuning Mao, Zacharie Delpierre Coudert, Zheng Yan, Zhengxing Chen, Zoe Papakipos, Aaditya Singh, Aayushi Srivastava, Abha Jain, Adam Kelsey, Adam Shajnfeld, Adithya Gangidi, Adolfo Victoria, Ahuva Goldstand, Ajay Menon, Ajay Sharma, Alex Boesenberg, Alexei Baevski, Allie Feinstein, Amanda Kallet, Amit Sangani, Amos Teo, Anam Yunus, Andrei Lupu, Andres Alvarado, Andrew Caples, Andrew Gu, Andrew Ho, Andrew Poulton, Andrew Ryan, Ankit Ramchandani, Annie Dong, Annie Franco, Anuj Goyal, Aparajita Saraf, Arkabandhu Chowdhury, Ashley Gabriel, Ashwin Bharambe, Assaf Eisenman, Azadeh Yazdan, Beau James, Ben Maurer, Benjamin Leonhardi, Bernie Huang, Beth Loyd, Beto De Paola, Bhargavi Paranjape, Bing Liu, Bo Wu, Boyu Ni, Braden Hancock, Bram Wasti, Brandon Spence, Brani Stojkovic, Brian Gamido, Britt Montalvo, Carl Parker, Carly Burton, Catalina Mejia, Ce Liu, Changhan Wang, Changkyu Kim, Chao Zhou, Chester Hu, Ching-Hsiang Chu, Chris Cai, Chris Tindal, Christoph Feichtenhofer, Cynthia Gao, Damon Civin, Dana Beaty, Daniel Kreymer, Daniel Li, David Adkins, David Xu, Davide Testuggine, Delia David, Devi Parikh, Diana Liskovich, Didem Foss, Dingkang Wang, Duc Le, Dustin Holland, Edward Dowling, Eissa Jamil, Elaine Montgomery, Eleonora Presani, Emily Hahn, Emily Wood, Eric-Tuan Le, Erik Brinkman, Esteban Arcaute, Evan Dunbar, Evan Smothers, Fei Sun, Fe-

lix Kreuk, Feng Tian, Filippos Kokkinos, Firat Ozgenel, Francesco Caggioni, Frank Kanayet, Frank Seide, Gabriela Medina Florez, Gabriella Schwarz, Gada Badeer, Georgia Swee, Gil Halpern, Grant Herman, Grigory Sizov, Guangyi, Zhang, Guna Lakshminarayanan, Hakan Inan, Hamid Shojanazeri, Han Zou, Hannah Wang, Hanwen Zha, Haroun Habeeb, Harrison Rudolph, Helen Suk, Henry Aspegren, Hunter Goldman, Hongyuan Zhan, Ibrahim Damlaj, Igor Molybog, Igor Tufanov, Ilias Leontiadis, Irina-Elena Veliche, Itai Gat, Jake Weissman, James Geboski, James Kohli, Janice Lam, Japhet Asher, Jean-Baptiste Gaya, Jeff Marcus, Jeff Tang, Jennifer Chan, Jenny Zhen, Jeremy Reizenstein, Jeremy Teboul, Jessica Zhong, Jian Jin, Jingyi Yang, Joe Cummings, Jon Carvill, Jon Shepard, Jonathan McPhie, Jonathan Torres, Josh Ginsburg, Junjie Wang, Kai Wu, Kam Hou U, Karan Saxena, Kartikay Khandelwal, Katayoun Zand, Kathy Matosich, Kaushik Veeraraghavan, Kelly Michelena, Keqian Li, Kiran Jagadeesh, Kun Huang, Kunal Chawla, Kyle Huang, Lailin Chen, Lakshya Garg, Lavender A, Leandro Silva, Lee Bell, Lei Zhang, Liangpeng Guo, Licheng Yu, Liron Moshkovich, Luca Wehrstedt, Madian Khabsa, Manav Avalani, Manish Bhatt, Martynas Mankus, Matan Hasson, Matthew Lennie, Matthias Reso, Maxim Groshev, Maxim Naumov, Maya Lathi, Meghan Keneally, Miao Liu, Michael L. Seltzer, Michal Valko, Michelle Restrepo, Mihir Patel, Mik Vyatskov, Mikayel Samvelyan, Mike Clark, Mike Macey, Mike Wang, Miquel Jubert Hermoso, Mo Metanat, Mohammad Rastegari, Munish Bansal, Nandhini Santhanam, Natascha Parks, Natasha White, Navyata Bawa, Nayan Singhal, Nick Egebo, Nicolas Usunier, Nikhil Mehta, Nikolay Pavlovich Laptev, Ning Dong, Norman Cheng, Oleg Chernoguz, Olivia Hart, Omkar Salpekar, Ozlem Kalinli, Parkin Kent, Parth Parekh, Paul Saab, Pavan Balaji, Pedro Rittner, Philip Bontrager, Pierre Roux, Piotr Dollar, Polina Zvyagina, Prashant Ratanchandani, Pritish Yuvraj, Qian Liang, Rachad Alao, Rachel Rodriguez, Rafi Ayub, Raghotham Murthy, Raghu Nayani, Rahul Mitra, Rangaprabhu Parthasarathy, Raymond Li, Rebekkah Hogan, Robin Battey, Rocky Wang, Russ Howes, Ruty Rinott, Sachin Mehta, Sachin Siby, Sai Jayesh Bondu, Samyak Datta, Sara Chugh, Sara Hunt, Sargun Dhillon, Sasha Sidorov, Satadru Pan, Saurabh Mahajan, Saurabh Verma, Seiji Yamamoto, Sharadh Ramaswamy, Shaun Lindsay, Shaun Lindsay, Sheng Feng, Shenghao Lin, Shengxin Cindy Zha, Shishir Patil, Shiva Shankar, Shuqiang Zhang, Shuqiang Zhang, Sinong Wang, Sneha Agarwal, Soji Sajuyigbe, Soumith Chintala, Stephanie Max, Stephen Chen, Steve Kehoe, Steve Satterfield, Sudarshan Govindaprasad, Sumit Gupta, Summer Deng, Sungmin Cho, Sunny Virk, Suraj Subramanian, Sy Choudhury, Sydney Goldman, Tal Remez, Tamar Glaser, Tamara Best, Thilo Koehler, Thomas Robinson, Tianhe Li, Tianjun Zhang, Tim Matthews, Timothy Chou, Tzook Shaked, Varun Vontimitta, Victoria Ajayi, Victoria Montanez, Vijai Mohan, Vinay Satish Kumar, Vishal Mangla, Vlad Ionescu, Vlad Poenaru, Vlad Tiberiu Mihailescu, Vladimir Ivanov, Wei Li, Wenchen Wang, Wenwen Jiang, Wes Bouaziz, Will Constable, Xiaocheng Tang, Xiaojian Wu, Xiaolan Wang, Xilun Wu, Xinbo Gao, Yaniv Kleinman, Yanjun Chen, Ye Hu, Ye Jia, Ye Qi, Yenda Li, Yilin Zhang, Ying Zhang, Yossi Adi, Youngjin Nam, Yu, Wang, Yu Zhao, Yuchen Hao, Yundi Qian, Yunlu Li, Yuzi He, Zach Rait, Zachary DeVito, Zef Rosnbrick, Zhaoduo Wen, Zhenyu Yang, Zhiwei Zhao, and Zhiyu Ma. 2024. The llama 3 herd of models. *Preprint*, arXiv:2407.21783.

[5] Xinyi Hou, Yanjie Zhao, Yue Liu, Zhou Yang, Kailong Wang, Li Li, Xiapu Luo, David Lo, John Grundy, and Haoyu Wang. 2024. Large language models for software engineering: A systematic literature review. *Preprint*, arXiv:2308.10620.

[6] Dong Huang, Jie M. Zhang, Michael Luck, Qingwen Bu, Yuhao Qing, and Heming Cui. 2024. Agentcoder: Multi-agent-based code generation with iterative testing and optimisation. *Preprint*, arXiv:2312.13010.

[7] Yiming Huang, Zhenghao Lin, Xiao Liu, Yeyun Gong, Shuai Lu, Fangyu Lei, Yaobo Liang, Yelong Shen, Chen Lin, Nan Duan, and Weizhu Chen. 2024. Competition-level problems are effective llm evaluators. *Preprint*, arXiv:2312.02143.

[8] Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. 2024. Swe-bench: Can language models resolve real-world github issues? *Preprint*, arXiv:2310.06770.

[9] 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. *CoRR*, abs/2001.08361.

[10] Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke Zettlemoyer, Percy Liang, Emmanuel Candès, and Tatsunori Hashimoto. 2025. s1: Simple test-time scaling. *Preprint*, arXiv:2501.19393.

[11] OpenAI, :, Ahmed El-Kishky, Alexander Wei, Andre Saraiva, Borys Minaiev, Daniel Selsam, David Dohan, Francis Song, Hunter Lightman, Ignasi Clavera, Jakub Pachocki, Jerry Tworek, Lorenz Kuhn, Lukasz Kaiser, Mark Chen, Max Schwarzer, Mostafa Rohaninejad, Nat McAleese, o3 contributors, Oleg Mürk, Rhythm Garg, Rui Shu, Szymon Sidor, Vineet Kosaraju, and Wenda Zhou. 2025. Competitive programming with large reasoning models. *Preprint*, arXiv:2502.06807.

[12] OpenAI, :, Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, Alex Iftimie, Alex Karpenko, Alex Tachard Passos, Alexander Neitz, Alexander

Prokofiev, Alexander Wei, Allison Tam, Ally Bennett, Ananya Kumar, Andre Saraiva, Andrea Vallone, Andrew Duberstein, Andrew Kondrich, Andrey Mishchenko, Andy Applebaum, Angela Jiang, Ashvin Nair, Barret Zoph, Behrooz Ghorbani, Ben Rossen, Benjamin Sokolowsky, Boaz Barak, Bob McGrew, Borys Minaiev, Botao Hao, Bowen Baker, Brandon Houghton, Brandon McKinzie, Brydon Eastman, Camillo Lugaresi, Cary Bassin, Cary Hudson, Chak Ming Li, Charles de Bourcy, Chelsea Voss, Chen Shen, Chong Zhang, Chris Koch, Chris Orsinger, Christopher Hesse, Claudia Fischer, Clive Chan, Dan Roberts, Daniel Kappler, Daniel Levy, Daniel Selsam, David Dohan, David Farhi, David Mely, David Robinson, Dimitris Tsipras, Doug Li, Dragos Oprica, Eben Freeman, Eddie Zhang, Edmund Wong, Elizabeth Proehl, Enoch Cheung, Eric Mitchell, Eric Wallace, Erik Ritter, Evan Mays, Fan Wang, Felipe Petroski Such, Filippo Raso, Florencia Leoni, Foivos Tsimpourlas, Francis Song, Fred von Lohmann, Freddie Sulit, Geoff Salmon, Giambattista Parascandolo, Gildas Chabot, Grace Zhao, Greg Brockman, Guillaume Leclerc, Hadi Salman, Haiming Bao, Hao Sheng, Hart Andrin, Hessam Bagherinezhad, Hongyu Ren, Hunter Lightman, Hyung Won Chung, Ian Kivlichan, Ian O'Connell, Ian Osband, Ignasi Clavera Gilaberte, Ilge Akkaya, Ilya Kostrikov, Ilya Sutskever, Irina Kofman, Jakub Pachocki, James Lennon, Jason Wei, Jean Harb, Jerry Twore, Jiacheng Feng, Jiahui Yu, Jiayi Weng, Jie Tang, Jieqi Yu, Joaquin Quiñonero Candela, Joe Palermo, Joel Parish, Johannes Heidecke, John Hallman, John Rizzo, Jonathan Gordon, Jonathan Uesato, Jonathan Ward, Joost Huizinga, Julie Wang, Kai Chen, Kai Xiao, Karan Singhal, Karina Nguyen, Karl Cobbe, Katy Shi, Kayla Wood, Kendra Rimbach, Keren Gu-Lemberg, Kevin Liu, Kevin Lu, Kevin Stone, Kevin Yu, Lama Ahmad, Lauren Yang, Leo Liu, Leon Maksin, Leyton Ho, Liam Fedus, Lilian Weng, Linden Li, Lindsay McCallum, Lindsey Held, Lorenz Kuhn, Lukas Kondraciuk, Lukasz Kaiser, Luke Metz, Madelaine Boyd, Maja Trebacz, Manas Joglekar, Mark Chen, Marko Tintor, Mason Meyer, Matt Jones, Matt Kaufer, Max Schwarzer, Meghan Shah, Mehmet Yatbaz, Melody Y. Guan, Mengyuan Xu, Mengyuan Yan, Mia Glaese, Mianna Chen, Michael Lampe, Michael Malek, Michele Wang, Michelle Fradin, Mike McClay, Mikhail Pavlov, Miles Wang, Mingxuan Wang, Mira Murati, Mo Bavarian, Mostafa Rohaninejad, Nat McAleese, Neil Chowdhury, Neil Chowdhury, Nick Ryder, Nikolas Tezak, Noam Brown, Ofir Nachum, Oleg Boiko, Oleg Murk, Olivia Watkins, Patrick Chao, Paul Ashbourne, Pavel Izmailov, Peter Zhokhov, Rachel Dias, Rahul Arora, Randall Lin, Rapha Gontijo Lopes, Raz Gaon, Reah Miyara, Reimar Leike, Renny Hwang, Rhythm Garg, Robin Brown, Roshan James, Rui Shu, Ryan Cheu, Ryan Greene, Saachi Jain, Sam Altman, Sam Toizer, Sam Toyer, Samuel Miserendino, Sandhini Agarwal, Santiago Hernandez, Sasha Baker, Scott McKinney, Scottie Yan, Shengjia Zhao, Shengli Hu, Shibani Santurkar, Shraman Ray Chaudhuri, Shuyuan Zhang, Siyuan Fu, Spencer Papay, Steph Lin, Suchir Balaji, Suvansh Sanjeev, Szymon Sidor, Tal Broda, Aidan Clark, Tao Wang, Taylor Gordon, Ted Sanders, Tejal Patwardhan, Thibault Sottiaux, Thomas Degry, Thomas Dimson, Tianhao Zheng, Timur Garipov, Tom Stasi, Trapit Bansal, Trevor Creech, Troy Peterson, Tyna Eloundou, Valerie Qi, Vineet Kosaraju, Vinnie Monaco, Vitchyr Pong, Vlad Fomenko, Weiyi Zheng, Wenda Zhou, Wes McCabe, Wojciech Zaremba, Yann Dubois, Yinghai Lu, Yining Chen, Young Cha, Yu Bai, Yuchen He, Yuchen Zhang, Yunyun Wang, Zheng Shao, and Zhuohan Li. 2024. Openai o1 system card. *Preprint*, arXiv:2412.16720.

[13] Shanghaoran Quan, Jiaxi Yang, Bowen Yu, Bo Zheng, Dayiheng Liu, An Yang, Xuancheng Ren, Bofei Gao, Yibo Miao, Yunlong Feng, Zekun Wang, Jian Yang, Zeyu Cui, Yang Fan, Yichang Zhang, Binyuan Hui, and Junyang Lin. 2025. Codeelo: Benchmarking competition-level code generation of llms with human-comparable elo ratings. *Preprint*, arXiv:2501.01257.

[14] Qwen, :, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tianyi Tang, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. 2025. Qwen2.5 technical report. *Preprint*, arXiv:2412.15115.

[15] Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *Preprint*, arXiv:2402.03300.

[16] Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. 2024. Scaling llm test-time compute optimally can be more effective than scaling model parameters. *Preprint*, arXiv:2408.03314.

[17] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael

Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. 2023. Llama 2: Open foundation and fine-tuned chat models. *Preprint*, arXiv:2307.09288.

[18] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. 2023. Chain-of-thought prompting elicits reasoning in large language models. *Preprint*, arXiv:2201.11903.

[19] Zhuoyan Xu, Zhenmei Shi, and Yingyu Liang. 2024. Do large language models have compositional ability? an investigation into limitations and scalability. *Preprint*, arXiv:2407.15720.

[20] John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. 2024. Swe-agent: Agent-computer interfaces enable automated software engineering. *Preprint*, arXiv:2405.15793.