AlphaCode 2 Technical Report

AlphaCode (Li et al., 2022) is the first artificial intelligence system to achieve an intermediate level in competitive programming, which involves complex reasoning tasks in advanced mathematics, logic, and computer science. This article introduces AlphaCode 2, a new system with significantly improved performance, powered by Gemini (Gemini team, Google, 2023). AlphaCode 2 relies on powerful language models and customized search and reordering mechanisms. When evaluated on the same platform as the original AlphaCode, we found that AlphaCode 2 solved 1.7 times the problem and performed better than 85% of the participants.

Intro

Competitive programming is one of the ultimate touchstones for testing programming skills . Participants are required to write code in a limited time to solve complex problems that require critical thinking, logic, and understanding of algorithms, coding, and natural-language. Therefore, it is an excellent benchmark for measuring advanced reasoning and problem-solving abilities.

AlphaCode (Li et al., 2022) is the first artificial intelligence system to achieve competitive level in this task. Its successor, AlphaCode 2, utilizes several models based on Gemini (Gemini team, Google, 2023) as part of a significantly improved system. When evaluated on the Codeforces platform, the mainstream platform for competitive programming, AlphaCode 2 solved 43% of the problems in 10 attempts, almost twice as many as the original AlphaCode (solving 25% of the problems). Although its predecessor performed among intermediate-level participants, we estimate that **AlphaCode 2 averaged 85th percentile** .

Adopting Gemini as the base model for all components of AlphaCode 2 was key to achieving this level of performance. The success of AlphaCode 2 highlights Gemini's flexibility and adaptability as we were able to fine-tune and optimize it to improve performance on several different tasks including code generation and code reordering.

Method

1. **Strategy model family** : This is a set of models that generate code samples for each problem.
2. **Sampling mechanism** : This mechanism encourages the generation of diverse code samples for searching in the possible program space.

3. **Filtering mechanism** : This mechanism is used to remove code samples that do not match the problem description.
4. **Clustering algorithm** : This algorithm groups semantically similar code samples to avoid redundancy.
5. **Scoring model** : This model is used to select the best candidate from every 10 largest code sample groups.

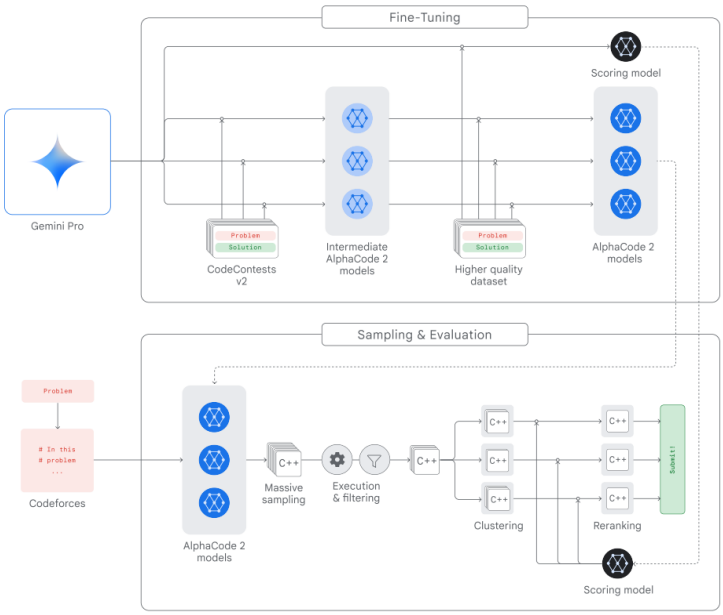


Figure 1 | High-level overview of the AlphaCode 2 system.

Policy and Fine-Tuning

Our starting point is the Gemini Pro model (Gemini team, Google, 2023), on which we applied two rounds of continuous fine-tuning, using GOLD (Pang and He, 2020) as the training target.

First, we fine-tuned the updated CodeContests dataset (which contains more questions, more solutions, and higher quality manually curated validation set tests). This dataset contains about 15,000 questions and 30 million code samples written by humans. We **generated multiple fine-tuned models by adjusting hyperparameters, and finally obtained a set of fine-tuned models** Second, we performed some additional fine-tuning steps on a different and higher quality dataset.

Relying on a set of strategies rather than a single strategy allows us to maximize diversity, which is critical to solving hard problems

Sampling

Our sampling method is similar to AlphaCode. For each problem, we generate up to one million code samples and use randomized temperature parameters for each sample to encourage

diversity. We also randomize the target metadata included in the prompt, such as the difficulty rating of the problem and its categorization label.

We evenly distribute the sampling budget across our family of fine-tuned models. While we sample Python and C++ in AlphaCode, for AlphaCode 2 we only use C++ samples because we find them to be of higher quality.

A large number of samples allow us to thoroughly search the model distribution and generate a large number of diverse code samples, maximizing the possibility of generating at least some correct samples. Given the number of samples, filtering and reordering are crucial for the performance of the entire system, as we only submit up to 10 code samples per problem.

Filtering

Each competitive programming problem contains at least one common input/output test, which indicates how the code samples should behave. We execute the corresponding test input for each code sample and filter out all samples that do not produce the expected output, as these samples cannot be correct, and also filter out less than 5% of the samples that cannot be compiled. On average, this filtering process removes about 95% of the samples.

Clustering

After filtering, we have an average of about 50,000 candidate solutions left for each problem, but we limit ourselves to submitting only 10. To further reduce the number of candidates, we aggregate samples based on their runtime behavior: like AlphaCode, we train a separate model to generate new test inputs for each problem, and then execute the remaining samples on these new inputs. The generated output forms a signature, which we use to group similar code samples into clusters. Then we sort them based on the cardinality of the cluster (i.e. the number of samples in the cluster) and only retain the largest 10 clusters.

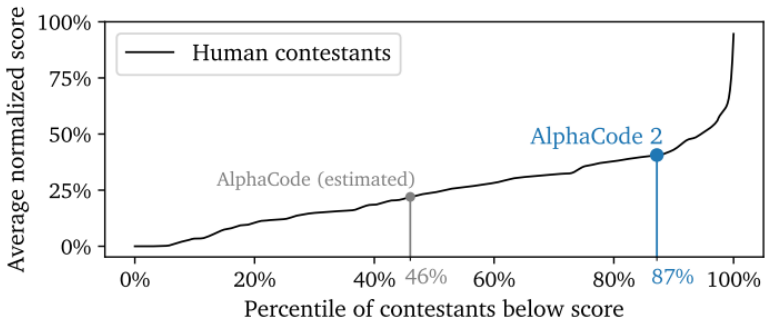
The purpose of clustering is to avoid redundancy: since code samples in the same cluster behave similarly, we can submit one sample from each cluster to the online evaluation system to obtain the best results. This way, we can filter out the most likely correct code from a large number of candidate solutions, improving the efficiency and success rate of problem-solving.

Scoring Model

We fine-tune the second Gemini Pro model to assign an estimated correctness score between 0 and 1 to the code samples. Using this scoring model, we calculate a score for each code sample in the remaining clusters; then we select the best candidate samples from each cluster based on this predicted score to form our final 10 submitted samples.

Evaluation

We evaluated AlphaCode 2 on the Codeforces platform, which is the same platform as the original AlphaCode. We selected 12 recent contests with over 8000 participants from the second level or the harder "1 + 2" level. This included a total of 77 questions. **For each question, we sampled one million candidate solutions and selected and sorted up to 10 solutions for submission according to the procedure detailed above until a correct solution was found or we exhausted all candidate solutions. (The final pass rate obtained from one million sampling is costly)**



We evaluated the impact of increasing the number of samples for each problem on performance. Similar to the situation with AlphaCode, we found that performance increases roughly logarithmically as the number of samples increases. AlphaCode 2 requires about 100 samples to achieve the performance level of AlphaCode when using one million samples, which improves its sample efficiency by more than 10,000 times.

Discussion

Competitive programming is very different from other programming tasks, which often follow an imperative paradigm: the user specifies clear instructions, and the model outputs the required code. In contrast, **competitive programming problems are often open-ended** . To solve these problems, problems need to be understood, analyzed, and reasoned about before code implementation is written, which involves concepts from advanced mathematics and computer science. (**Strong foundational models are required**)

Using Gemini Pro as our base model significantly improves the performance of two key components of the system: **a policy model for generating code samples, and a scoring model for selecting the best samples** We were able to fine-tune Gemini to the high performance of these two very different tasks, which demonstrates its amazing flexibility. We speculate that if we use Gemini Ultra, which has improved coding and inference capabilities, as the base model, it will lead to further improvements in the overall AlphaCode 2 method.

Although AlphaCode 2 has achieved impressive results, there is still a lot of work to be done before we can see a system that can reliably achieve the best human coder performance. **Our**

system requires a lot of trial and error, and is still too expensive to run on a large scale. In addition, it relies heavily on code samples that can filter out obvious errors .

This opens the door to positive interaction between the system and human coders, **Human coders can specify additional filtering attributes** ; In this AlphaCode 2-human combination setup, we scored over the 90th percentile! We hope that this interactive programming will be the future of programming, where programmers can use highly capable AI models as collaborative tools to help them reason through problems, propose code designs, and assist in implementation. We are working to introduce the unique capabilities of AlphaCode 2 into our foundational Gemini model as the first step in making this new programming paradigm available to everyone.