2023-12-06

AlphaCode 2 Technical Report

AlphaCode Team, Google DeepMind

AlphaCode (Li et al., 2022) was the first AI system to perform at the level of the median competitor in competitive programming, a difficult reasoning task involving advanced maths, logic and computer science. This paper introduces AlphaCode 2, a new and enhanced system with massively improved performance, powered by Gemini (Gemini Team, Google, 2023). AlphaCode 2 relies on the combination of powerful language models and a bespoke search and reranking mechanism. When evaluated on the same platform as the original AlphaCode, we found that AlphaCode 2 solved $1.7\times$ more problems, and performed better than 85% of competition participants.

Introduction

Competitive programming is one of the ultimate litmus tests of coding skills. Participants are given limited time to write code to solve complex problems that require critical thinking, logic, and understanding of algorithms, coding and natural language. As such, it is a great benchmark for advanced reasoning and problem solving abilities.

AlphaCode (Li et al., 2022) was the first AI system to reach a competitive level on this task. Its successor, AlphaCode 2, leverages several Gemini-based (Gemini Team, Google, 2023) models as part of a massively improved system. When evaluated on the Codeforces platform – a mainstay of competitive programming – AlphaCode 2 solves 43% of problems within 10 attempts, close to twice as many problems as the original AlphaCode (25%). While its predecessor performed at the level of the median competitor, we estimate that AlphaCode 2 reaches the 85th percentile on average.

Adopting Gemini as the foundation model for all components of AlphaCode 2 was key to achieving this level of performance. AlphaCode 2's success underlines Gemini's flexibility and adaptability, as we were able to fine-tune it and optimize performance for several different tasks including code generation and code reranking.

Overall System

AlphaCode 2 relies on powerful Large Language Models, combined with an advanced search and reranking mechanism tailored for competitive programming. As detailed in Figure 1, its main components include:

- A family of policy models which generate code samples for each problem;
- A sampling mechanism that encourages generating a wide diversity of code samples to search over the space of possible programs;
- A filtering mechanism to remove code samples that do not comply with the problem description;
- A clustering algorithm that groups semantically similar code samples, allowing us to avoid redundancies;
- A scoring model which we use to surface the best candidate out of each of the 10 biggest code samples clusters.

Corresponding author(s): remileblond@google.com

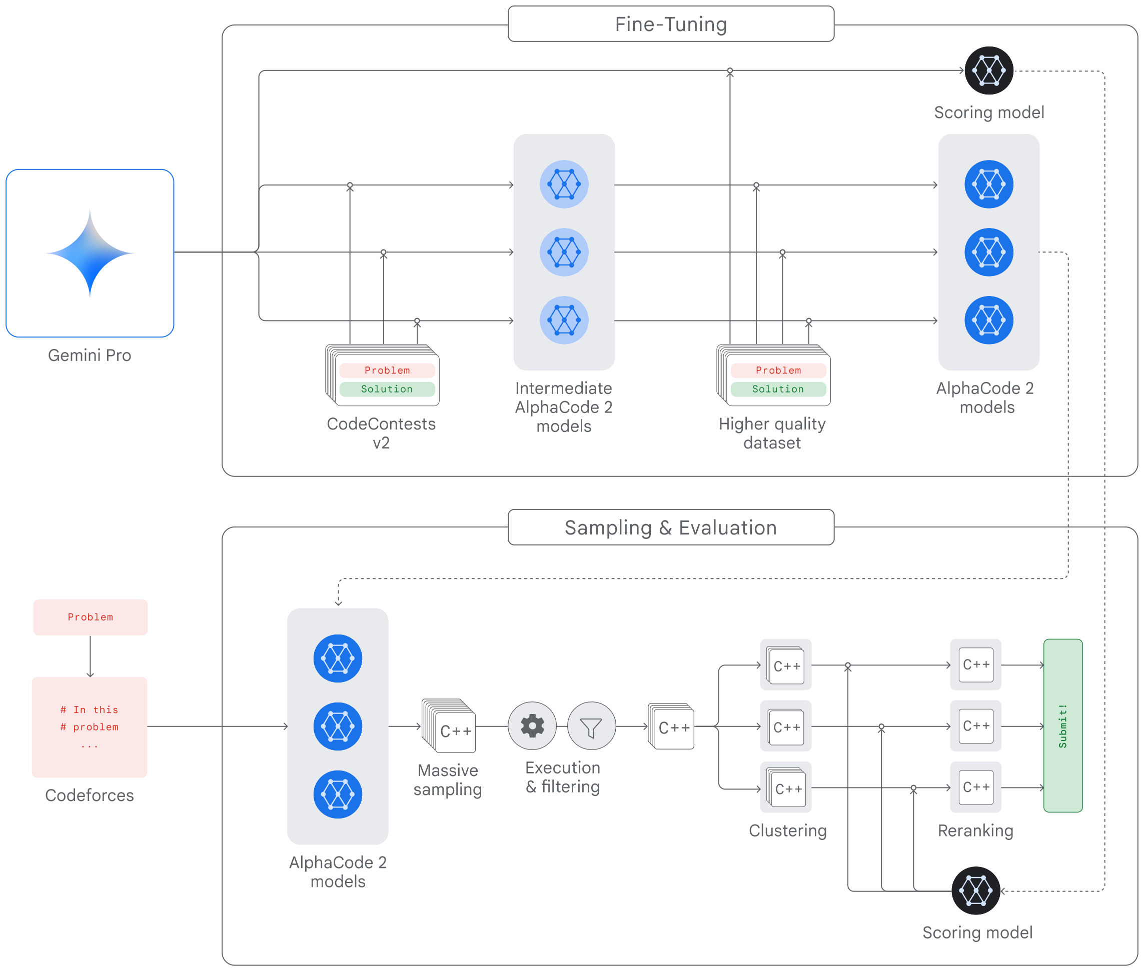

Figure 1 | High-level overview of the AlphaCode 2 system.

Policy and Fine-Tuning

Our starting point is the Gemini Pro model (Gemini Team, Google, 2023), on which we apply two consecutive rounds of fine-tuning using GOLD (Pang and He, 2020) as the training objective.

First, we fine-tune on an updated version of the CodeContests dataset (containing more problems, more solutions and higher quality, manually-curated tests on the validation set). This dataset contains approximately 15 thousand problems and 30 million human code samples. We generate several fine-tuned models by varying hyperparameters, and end up with a family of fine-tuned models. Second, we conduct a few additional steps of fine-tuning on a different, higher-quality dataset.

Relying on a family of policies instead of a single one allows us to maximize diversity, which remains key to tackling hard problems.

Sampling

Our sampling approach is close to that of AlphaCode. We generate up to a million code samples per problem, using a randomized temperature parameter for each sample to encourage diversity. We also randomize targeted metadata included in the prompt, such as the problem difficulty rating and its categorical tags.

We split our sampling budget evenly across our family of fine-tuned models. While we sampled in Python and C++ for AlphaCode, we only used C++ samples for AlphaCode 2 as we found them to be higher quality.

Massive sampling allows us to search the model distribution thoroughly and generate a large diversity of code samples, maximizing the likelihood of generating at least some correct samples. Given the amount of samples, filtering and reranking are of paramount importance to the overall system's performance, as we only submit a maximum of 10 code samples per problem.

Filtering

Each competitive programming problem contains at least one public input/output test indicating how code samples should behave. We execute each code sample on the corresponding test input, and filter out all which do not produce the expected output and therefore could not have been correct, as well as the less than 5% of samples that do not compile. On average, this filtering removes approximately 95% of the samples.

Clustering

After filtering, we are left with an average of 50 thousand candidates per problem, but we limit ourselves to 10 submissions. To further trim down candidates, we aggregate samples based on their runtime behavior: as in AlphaCode, we train a separate model to generate new test inputs for each problem, then execute the remaining samples on these new inputs. The produced outputs form a signature that we use to group similar code samples together into clusters. We then order the clusters according to their cardinality, and only keep the 10 largest.

The point of clustering is to avoid redundancies: since code samples in the same cluster behave similarly, we can submit a single one per cluster to the online judge to obtain the best result.

Scoring Model

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

Evaluation

We evaluated AlphaCode 2 on Codeforces, the same platform as the original AlphaCode. We selected 12 recent contests with more than 8000 participants, either from division 2 or the harder division "1+2". This makes for a total of 77 problems. For each problem, we sampled one million candidates and submitted up to 10 solutions selected and ordered according to the procedure detailed above, until either one correct solution was found, or we ran out of candidates.

We found AlphaCode 2 solved 43% of these competition problems, a close to 2× improvement over the prior record-setting AlphaCode system, which solved 25%. Mapping this to competition rankings, we estimate that AlphaCode 2 sits at the 85th percentile on average – i.e. it performs better than 85% of entrants, ranking just between the 'Expert' and 'Candidate Master' categories on Codeforces. This is

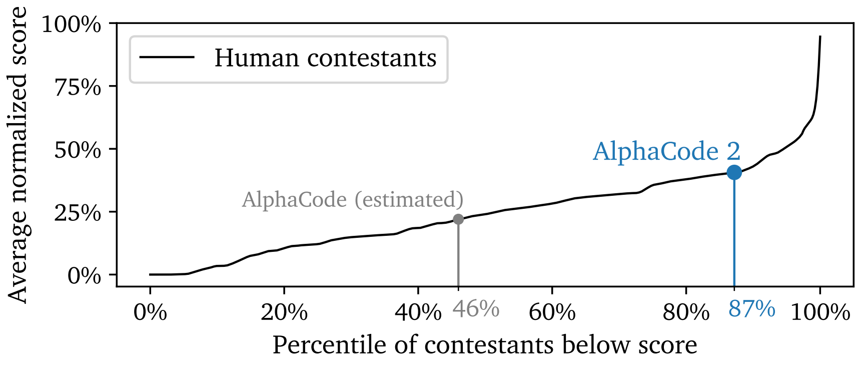

Figure 2 | Estimated ranking of AlphaCode 2. We plot the Codeforces score of human competitors – normalized to [0, 1] by dividing by the best human score per contest – against their ranking, averaged on the 12 contests we evaluate on. We then compute AlphaCode 2's average normalized score and report it on the ranking axis, which puts it comfortably above the 85th percentile. This ranking accounts for a simulated time penalty assuming that AlphaCode 2 goes through the problems by increasing difficulty, and finishes sampling for the last problem at the 2 hour mark.

a significant advance over AlphaCode, which only outperformed an estimated 46% of competitors. In the two contests where it performs best, AlphaCode 2 outperforms more than 99.5% of competition participants!

We evaluated the impact of increasing the amount of samples per problem. As was the case for AlphaCode, we find that performance increases roughly log-linearly with more samples. AlphaCode 2 requires about 100 samples to reach the level of performance of AlphaCode with a million samples, making it over 10000× more sample efficient.

Discussion and Conclusion

Competitive programming is very different from other coding tasks, which usually follow the imperative paradigm: the user specifies clear instructions and the model outputs the required code. In contrast, competitive programming problems are open-ended. To solve them, before writing the code implementation one needs to understand, analyze and reason about the problem, which involves advanced mathematics and computer science notions.

This explains why generally-available AI systems perform poorly on this benchmark. AlphaCode 2's success on competitive programming contests represents an impressive step change in performance on this extremely hard reasoning task.

Adopting Gemini Pro as our foundation model led to significant increases in performance on two critical components of the system: the policy models generating the code samples, and the scoring model used to select the best of them. The fact that we were able to fine-tune Gemini to high performance for these two very different tasks speaks to its incredible flexibility. We suspect using Gemini Ultra as the foundation model instead, with its improved coding and reasoning capabilities, would lead to further improvements in the overall AlphaCode 2 approach.

Despite AlphaCode 2's impressive results, a lot more remains to be done before we see systems that can reliably reach the performance of the best human coders. Our system requires a lot of trial and error, and remains too costly to operate at scale. Further, it relies heavily on being able to filter out obviously bad code samples.

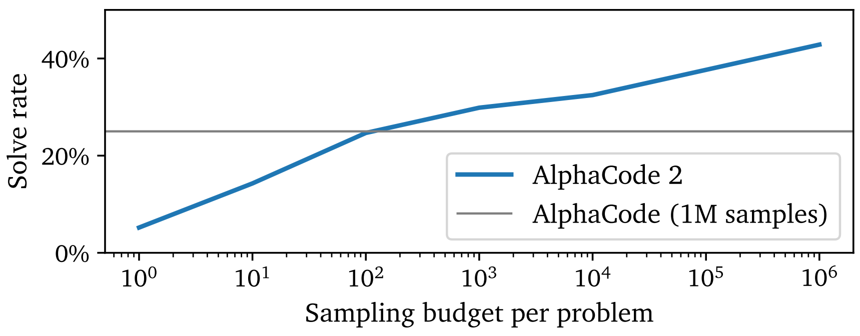

Figure 3 | Solve rate on 12 recent contests as a function of the number of samples per problem.

This opens the door for a positive interaction between the system and human coders, who can specify additional filtering properties; in this AlphaCode 2 + human setting, we score above the 90th percentile! We hope this kind of interactive coding will be the future of programming, where programmers make use of highly-capable AI models as collaborative tools that can help them reason about the problems, propose code designs, and assist with implementation. We are working towards bringing AlphaCode 2's unique capabilities to our foundation Gemini models as a first step to make this new programming paradigm available to everyone.

References

Gemini Team, Google. Gemini: A Family of Highly Capable Multimodal Models. 2023. URL https://storage.googleapis.com/deepmind-media/gemini/gemini_1_report.pdf.

Leblond et al. AlphaCode 2 Technical Report. 2023. URL https://storage.googleapis.com/deepmind-media/AlphaCode2/AlphaCode2_Tech_Report.pdf.

Y. Li, D. Choi, J. Chung, N. Kushman, J. Schrittwieser, R. Leblond, T. Eccles, J. Keeling, F. Gimeno, A. D. Lago, T. Hubert, P. Choy, C. de Masson d'Autume, I. Babuschkin, X. Chen, P.-S. Huang, J. Welbl, S. Gowal, A. Cherepanov, J. Molloy, D. J. Mankowitz, E. S. Robson, P. Kohli, N. de Freitas, K. Kavukcuoglu, and O. Vinyals. Competition-level code generation with alphacode. *Science*, 2022. URL https://www.science.org/doi/abs/10.1126/science.abq1158.

R. Y. Pang and H. He. Text generation by learning from demonstrations. *arXiv preprint arXiv:2009.07839*, 2020. URL https://arxiv.org/pdf/2009.07839.pdf.

Citing this work

This is a free, open access paper provided by Google DeepMind. The final version of this work was published online. *Cite as:*

Leblond et al. AlphaCode 2 Technical Report. 2023. URL https://storage.googleapis.com/deepmind-media/AlphaCode2/AlphaCode2_Tech_Report.pdf.

Contributions and Acknowledgments

Leads
Felix Gimeno, *technical lead*
Florent Altché, *technical lead*
Rémi Leblond, *lead*

Core contributors
Alaa Saade
Anton Ruddock
Corentin Tallec
George Powell
Jean-Bastien Grill
Maciej Mikuła
Matthias Lochbrunner
Michael Mathieu
Paul Caron

Contributors
Disha Shrivastava
Eric Mitchell
Grace Margand
Jacob Kelly
Jakub Sygnowski
James Keeling
Junyoung Chung
Nate Kushman
Nikolay Savinov
Petko Yotov
Tobenna Peter Igwe
Wojciech Stokowiec
Yujia Li

Strategic Advisors
Koray Kavukcuoglu
Nando de Freitas
Oriol Vinyals
Pushmeet Kohli
Satinder Baveja

Within each role, authors are listed alphabetically; ordering is not indicative of level of contributions.

We gratefully acknowledge the support from our colleagues at Google DeepMind, especially the Gemini team, the original AlphaCode team and the hardware team, without whom Alphacode 2 would not have been possible.

We are indebted to Eleanor Tomlison and Gaby Pearl for our very nice AlphaCode 2 system overview figure.

We would like to thank our leadership, reviewers and colleagues for valuable discussions and feedback on the report – Aakanksha Chowdhery, Aliya Ahmad, Antonia Paterson, Arielle Bier, Ben Bariach, Dawn Bloxwich, Demis Hassabis, Eli Collins, Emily Hossellman, Fred Alcober, Jeff Dean, Joel Moss, Jon Small, Koray Kavukcuoglu, Lily Lin, Megha Goel, Oriol Vinyals, Petar Veličković, Rebecca Bland, Sanah Choudhry and Tessa Lueth.

The problem used in the AlphaCode 2 demo – displayed on the main Gemini webpage – is used with the permission of Codeforces, a platform which hosts regular competitions of participants from around the world who come to test their coding skills. The problem is from the CodeTON Round 4 contest.