CASE STUDIES

# Case Study: How Our Team Won the Mega Chess Hackathon with Deep Learning and Rap Iteration

STRONG COMPUTE

SEP 25, 2024

Our team recently competed in and won the Strong Compute Mega Chess Hackathon. Ten San Francisco and Sydney teams competed simultaneously to build the strongest possible chess-playing deep learning model in just two day The event culminated in a model vs. model tournament, where the bots faced

to determine the final winner. It was an exciting and challenging experience. W would like to share some of the lessons learned along the way.

## Team Formation and Strategy

Our team comprised Justin F. Knoll, Suryaprakash Senthil Kumar, and Ashish Mukharji. We did not know each other before the event but made a point to connect via Zoom and share our backgrounds, possible technical approaches, working styles, and goals for the event beforehand. We were confident in our t and approach before the event started. Forming a team and getting a rough consensus on our approach before the event started saved us precious hacking time and is a highly recommended tactic.



Once the event began, we dove in to familiarize ourselves with the Strong Compute ISC platform, the provided datasets and models, and the actual tournament gameplay example scripts.

Our first objective was to close the loop: to train a very basic model from randomized weights into a candidate model competing in a one-round mock tournament. It's hard to overstate how valuable this was in ensuring we understood all parts of the stack, the submission requirements, and the tournament API.

We started a multi-hour training run and pulled one of the intermediate checkpoints to close the loop. Seeing even a very weak and undertrained mode playing chess on a live-refreshing board was a magical moment! The gameplay test script gave us a way to evaluate models against each other heuristically.

We let the model train and monitored training loss, rank correlation, adaptive learning rate adjustments, etc. to gauge training performance.

# Exploring a Range of Technical Approaches

Confident that we understood the full stack and submission requirements, and with a way to approximately evaluate model performance, we turned our attention to selecting our own moves as a team within the tournament.

Given the complexity of building a competitive chess-playing model, we explored two high-level approaches: a vision-like model and a GPT-based model. One key inspiration for the GPT-based models was **Adam Karvonen's paper** on "Emergent World Models and Latent Variable Estimation in Chess-Playing Language Models." Within the vision approach, we experimented with CNNs, transformers and multi-head attention.

Adam's mechanistic interpretability research on Chess-GPT models applied linear probes to analyze Chess-GPT network activations and concluded that the network creates an emergent world model including the chessboard, piece positions, and even latent variables like player ELO rating. Learning about this research was a fascinating side quest, but the mechanistic interpretability results were not ultimately about model performance, but about emergent world models.

At times, we worried about training a GPT-based model based on Leela Chess Zero self-play, since such a model is ultimately doing probabilistic next-token prediction over examples from the training corpus, and one presumes that some of the Leela Chess Zero self-play games from early training would be examples of spectacularly poor play! On the other hand, if one were to train a GPT-based model on only grandmaster games, it would never have seen the sort of blunders we expected to encounter in the tournament models, and so wouldn't know how to exploit them. In general, the GPT-based chess models are fascinating but seem harder to reason about how to train for high performance.

We did some initial hyperparameter tuning of the models by modifying values observing training metrics over short “cycle” test runs, then using the selected parameters for longer “burst” runs. We stopped short of doing a more structur hyperparameter sweep.

## Team-Level Divide and Conquer

We were able to divide and conquer by having individual team members focus optimizing different model approaches and run long-lived “burst” jobs to train those models in parallel. This was key to parallelizing progress and maximizing how quickly we could validate — or discard — our hypotheses.

We had access to multi-gigabyte datasets, including historical grandmaster ga and Leela Chess Zero self-play. We experimented with merging some of the provided datasets, which was not difficult and seemed effective. We experimen with adding our own outside data sets (for example, a 29GB Lichess database export), but time constraints forced us to focus more on model tuning and train on the provided datasets than on converting and ingesting outside data.

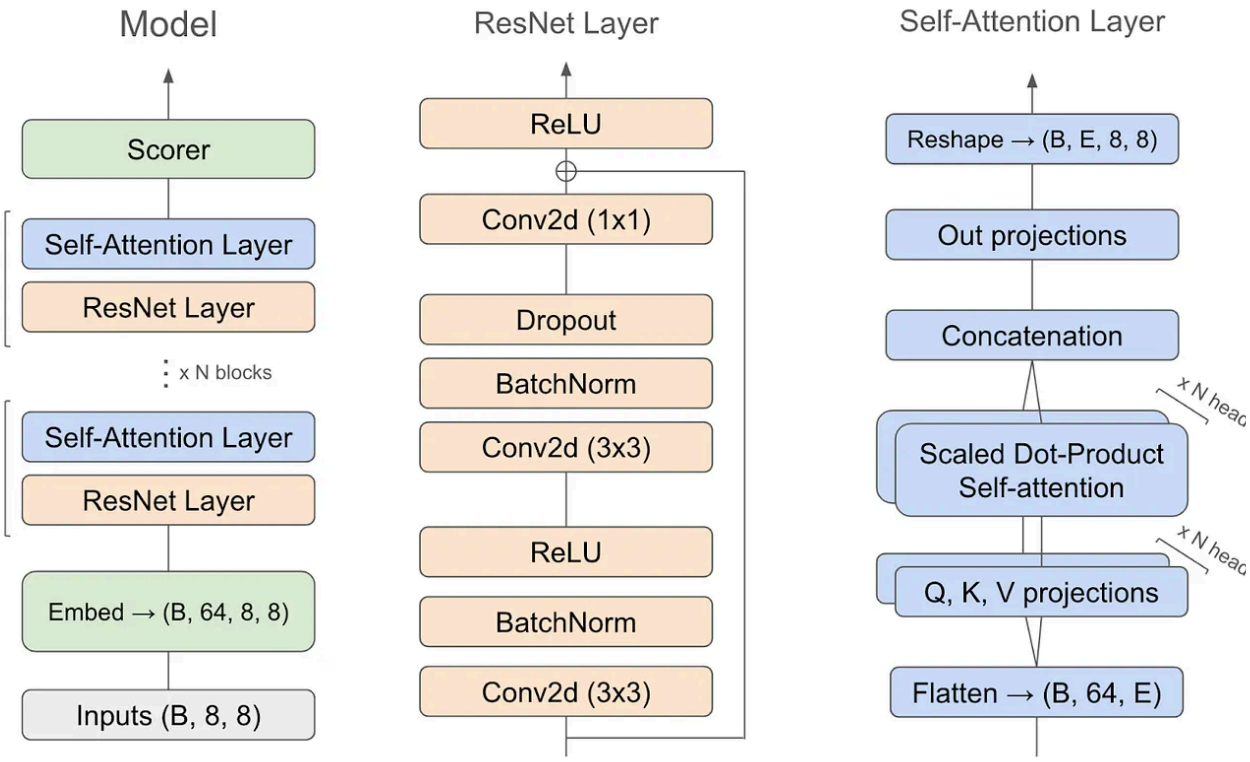
## Leveraging Strong Compute's ISC for Trainin

All teams were provided with access to Strong Compute's service, which made possible for us to train our models using powerful 72xA100 clusters. This infrastructure was a game-changer for rapid iteration.

## The Winning Model

On day two, we selected some of the longest-trained models with the best train metrics and played them against each other using the gameplay script, keeping informal tally of performance. The vision approaches were dominant, so we did some final tuning and played a sub-tournament amongst the strongest two vision models, ultimately selecting a CNN-based model with multi-head attention an

dilated convolution to expand the receptive field and capture relationships fur across the board.




# Post-Hackathon Reflection

Participating in this hackathon was a valuable learning experience. Two days not much time to build and coordinate as a team, and we didn't get a chance t implement many of our ideas. Just as with building a commercial product, we to strike a balance between speed and rigor and aim to maximize the rate of validated learning.

While overall time was scarce, the ability to easily do distributed training on a 72xA100 cluster was a game changer. More data, more epochs, deeper models: of these were feasible. Ensuring that we were always using the cluster for som experiment and not letting it idle was an important tactic.

# Acknowledgments

A huge thanks to my teammates and everyone who made this event possible, especially **Ben Sand, Adam Peaston, Tim Smoothy, and Rebecca Pham from Strong Compute**, for providing the infrastructure and support that allowed us compete at this level.

# Conclusion

This hackathon pushed our limits and taught us the value of rapid iteration, strategic model selection, and leveraging powerful computing resources. We'r thrilled with our success and glad to be able to share the lessons here.

This was written by Chess Hackathon participants, Ashish Mukharji, Justin F Knoll and Suryaprakash Senthil Kumar.

Try Strong Compute at our next event.

Write a comment...