Minigo

Making a Go AI using Tensorflow and Kubernetes

Andrew Jackson • Josh Hoak

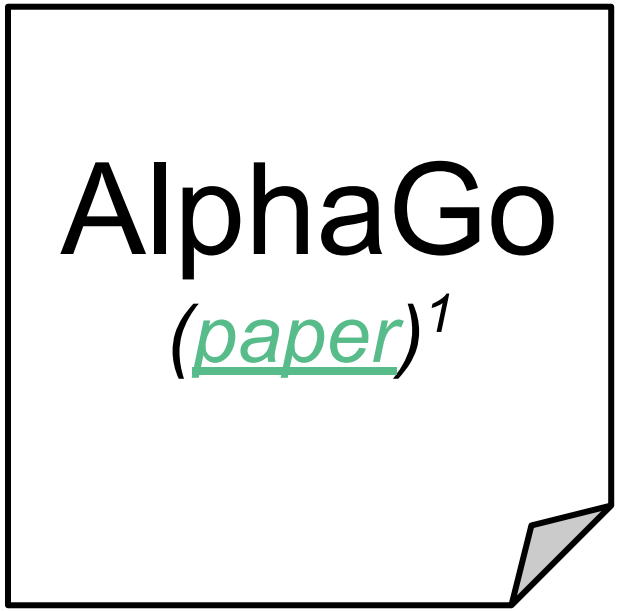
What is Minigo?

github.com/tensorflow/minigo

Why We Created Minigo

- We love Go!
- Go players want to study the games, learn new things
- Go still pushes the boundaries of Machine Learning and Computing
- Possible to replicate the results of AlphaGoZero?

***Note**: Not AlphaGo, not related to Deepmind in any way.*



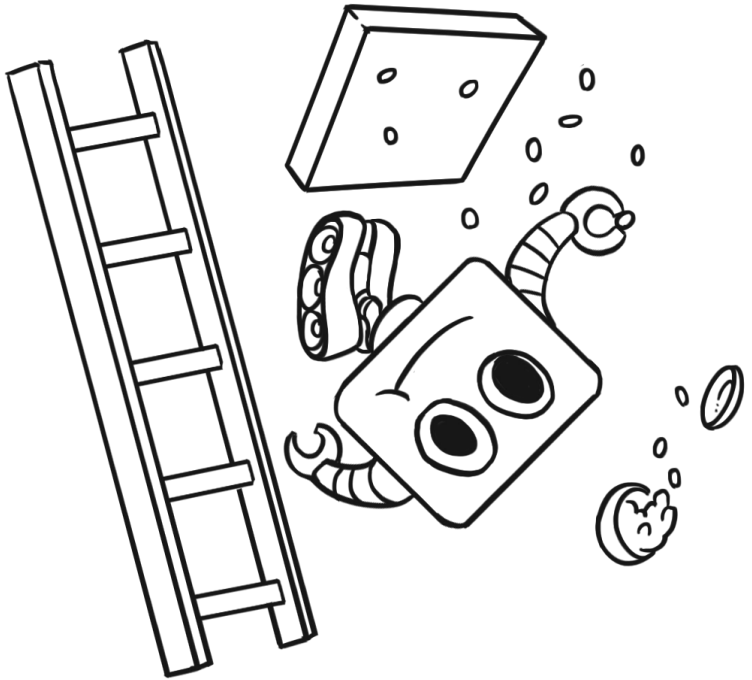
MuGo

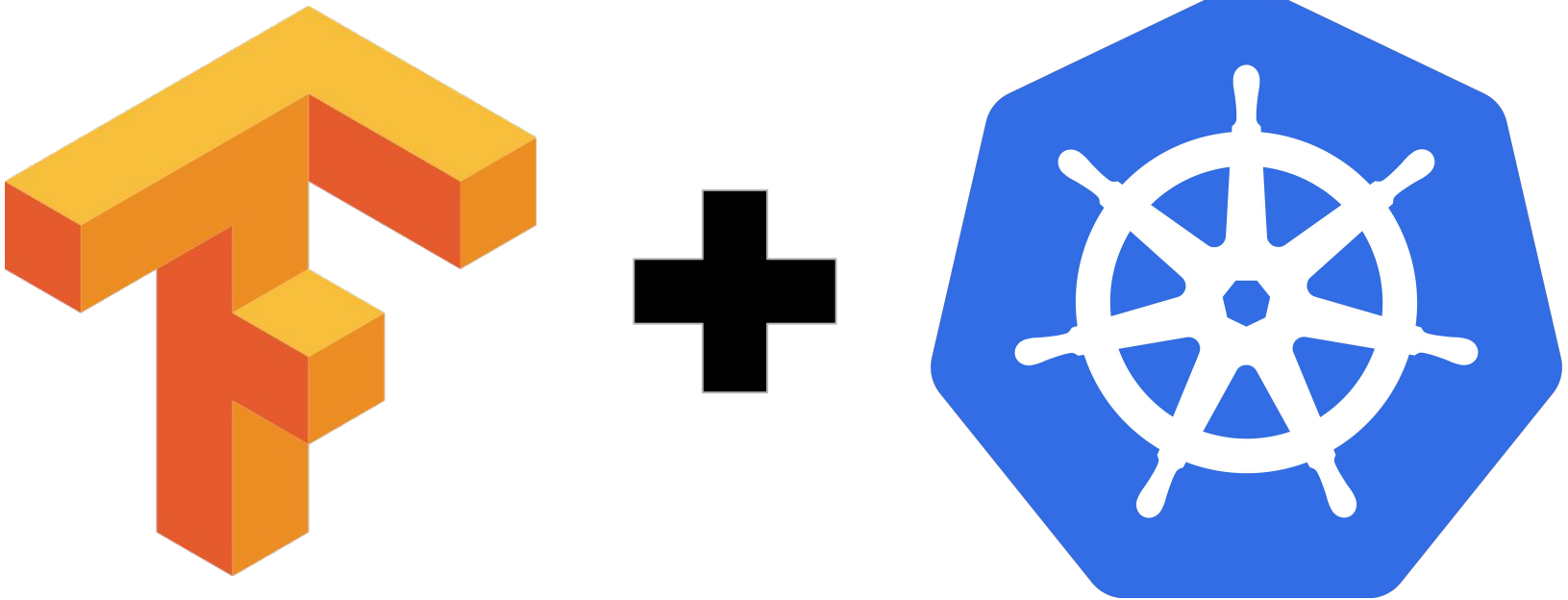
(github.com/brilee/MuGo)

[1]David Silver, et. al. Mastering the game of Go with deep neural networks and tree search. *Nature,* 529(7587):484–489, January 2016

[1]David Silver, et. al. Mastering the game of go without human knowledge. *Nature*, 550:354– 359, 2017.

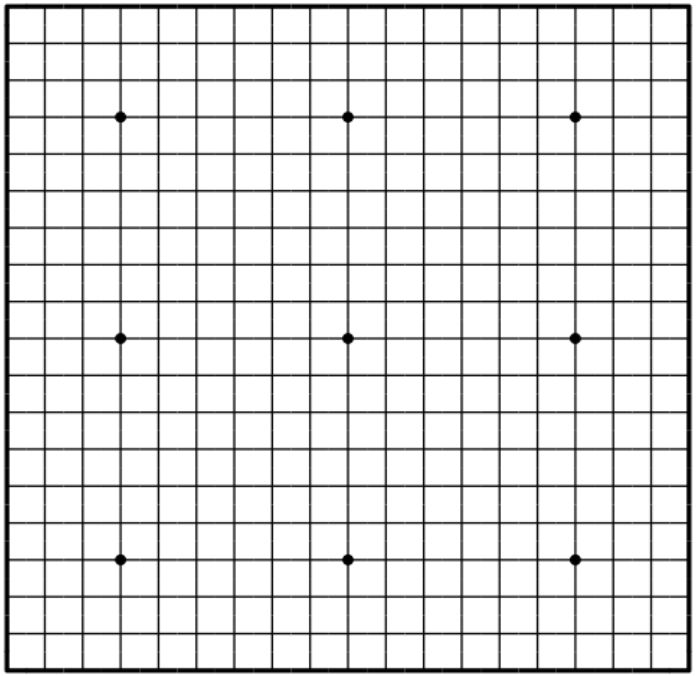
=

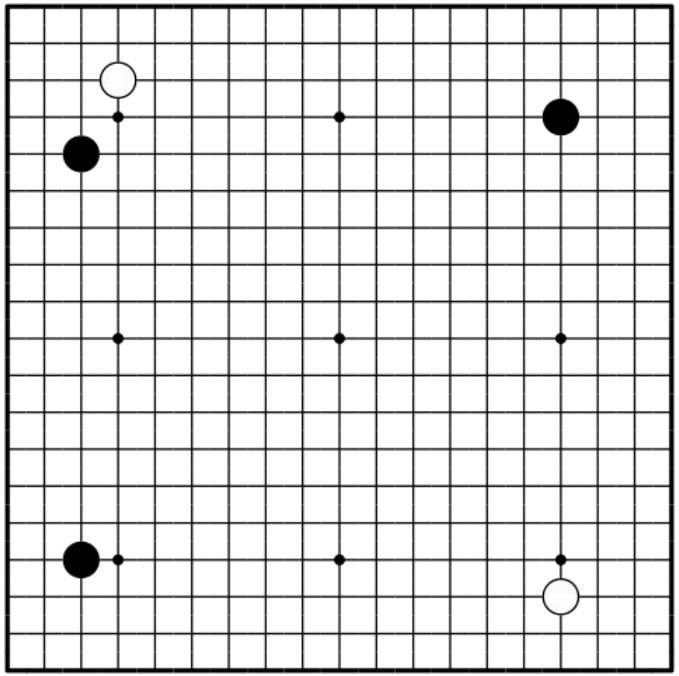




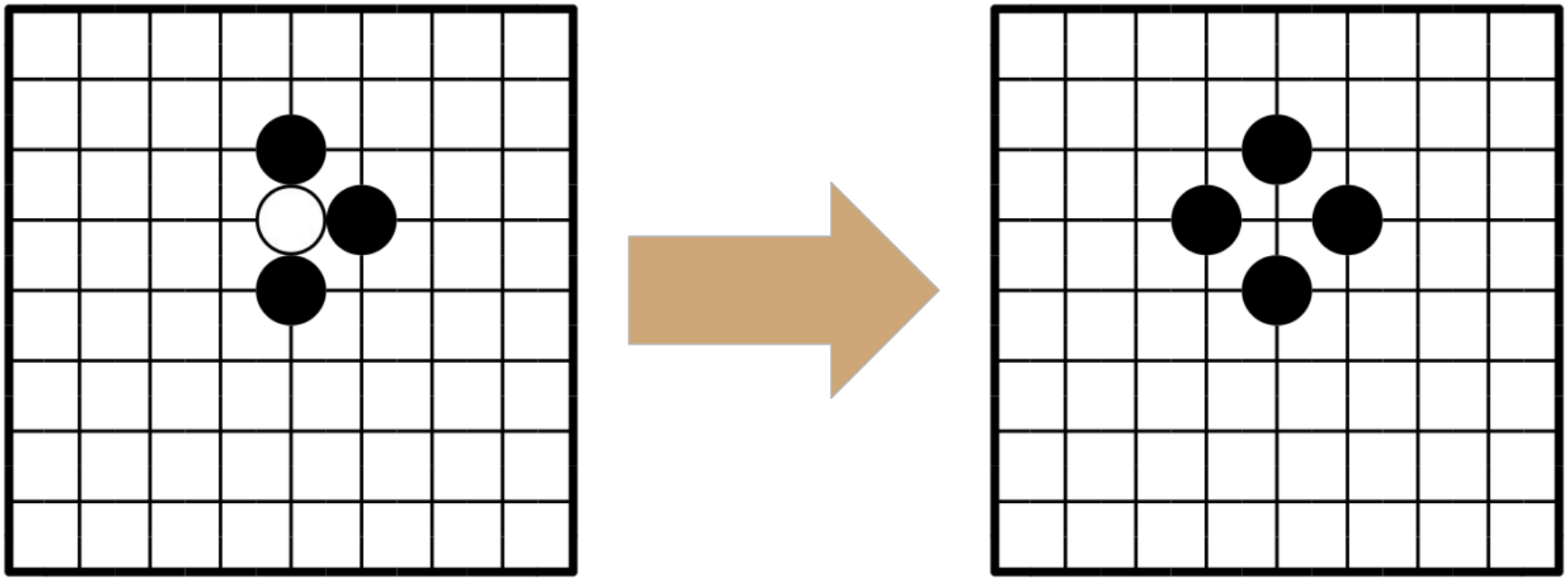
Go in Five Slides

Beginning a Game

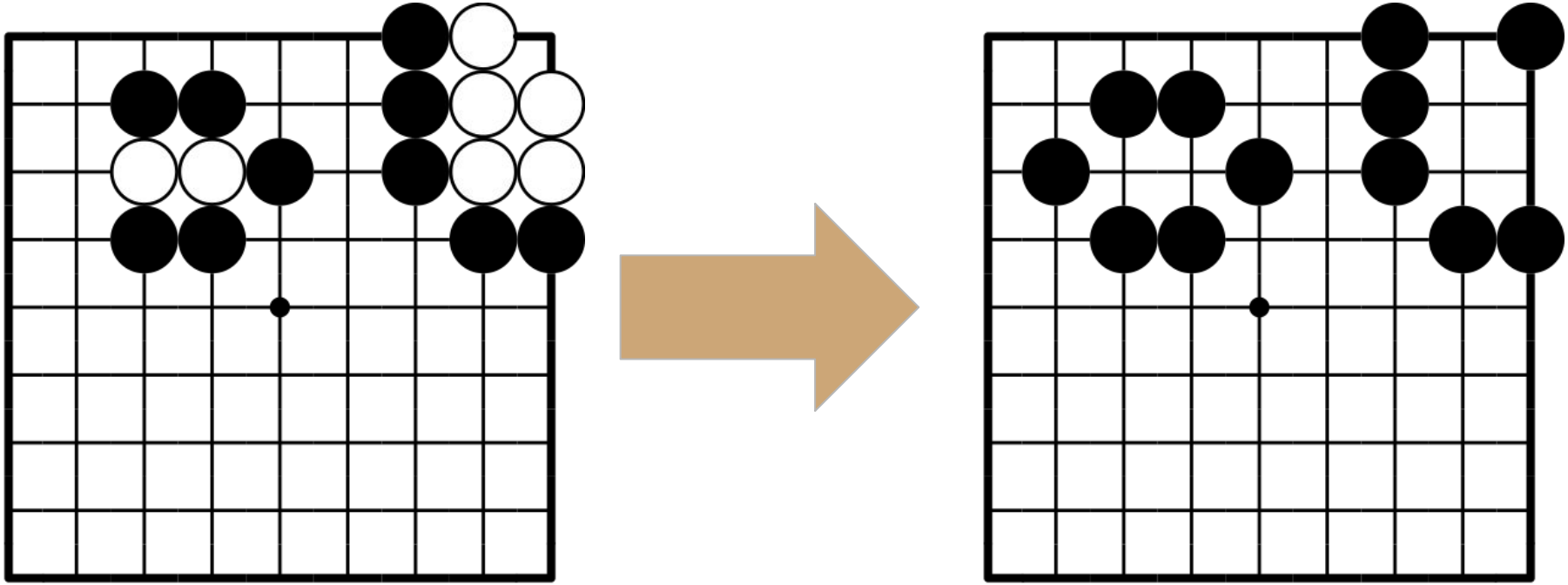




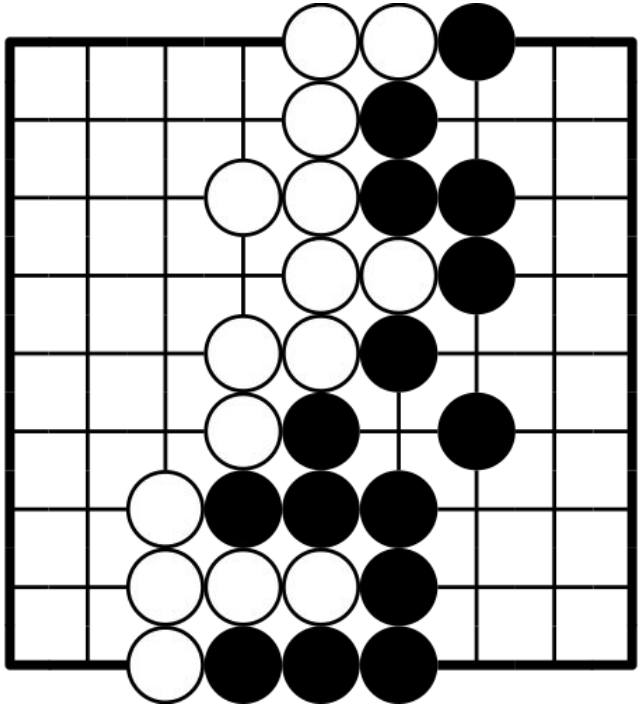
Capturing

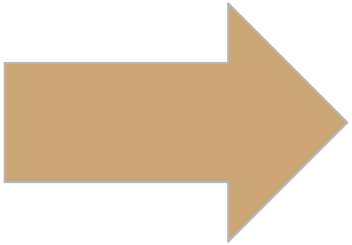


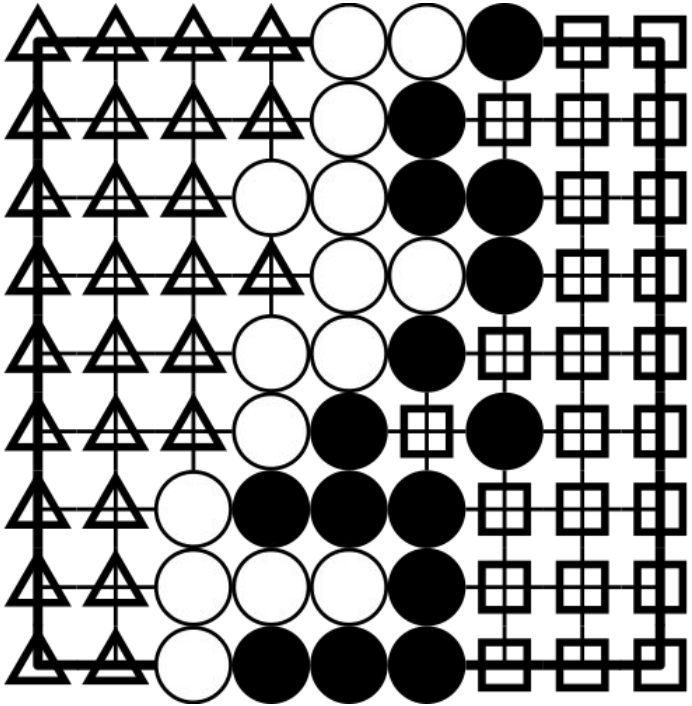
Capturing



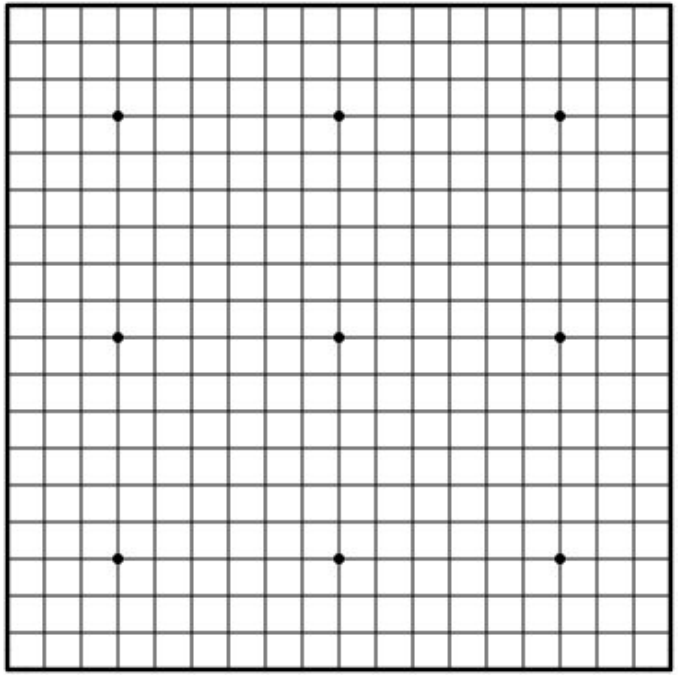
Points





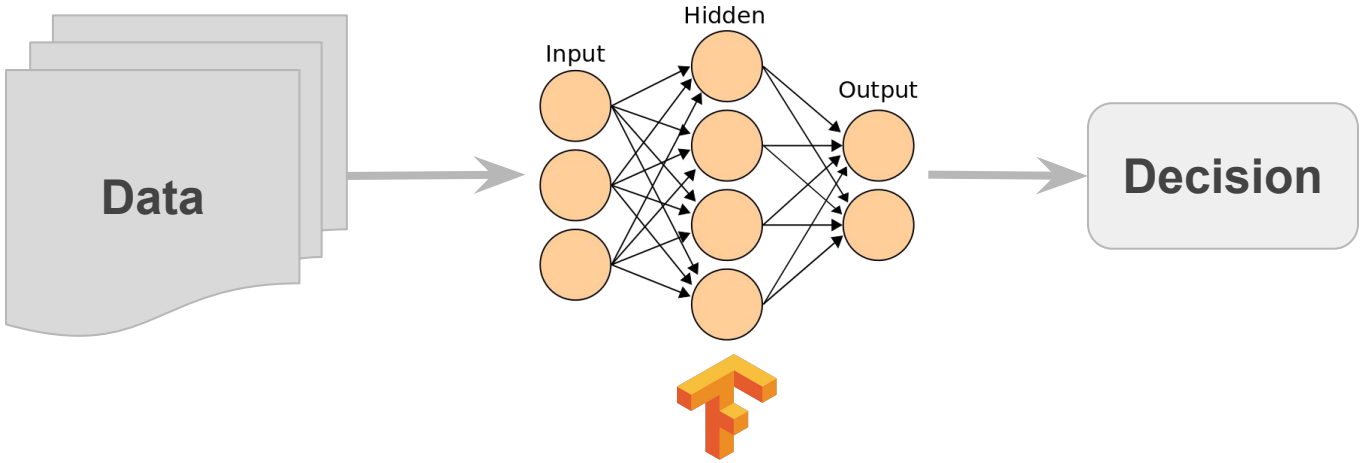


Sneak Peak: Minigo In Action



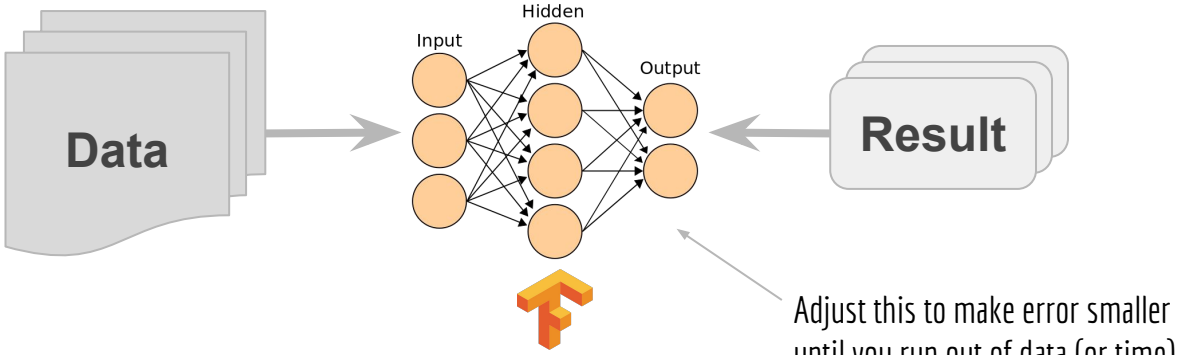
Machine Learning and Minigo (Also In Five Slides)

Neural Networks - Inference

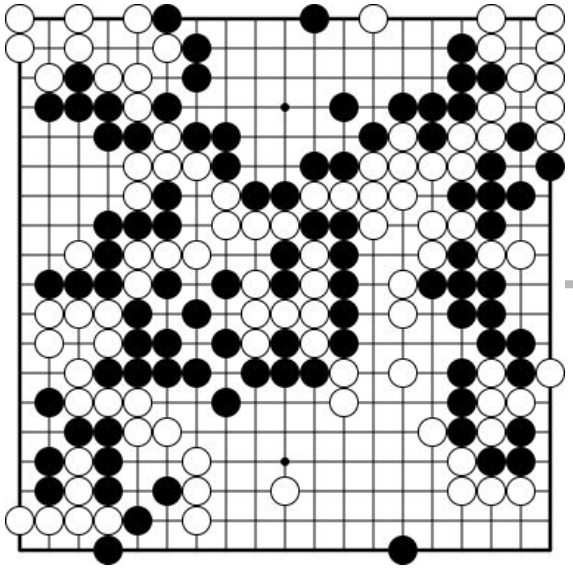


Neural Networks - Training

Quantify the “error” of your inference as a function:
Error = f(Data, Result)



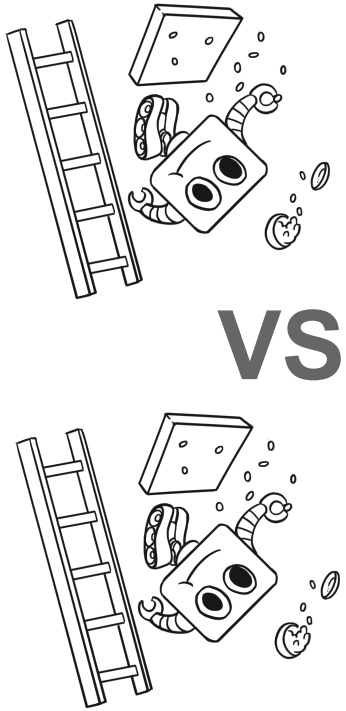
Inference for Minigo

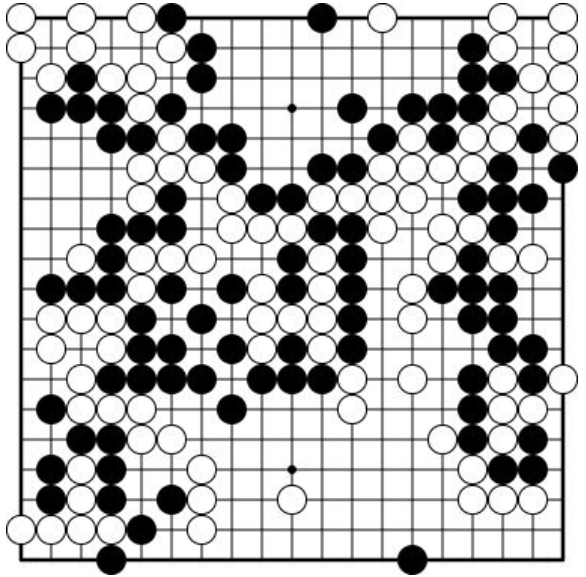


Which Move?

Who's Ahead?

Self-Play with Monte Carlo Tree Search





Minigo's "Error Function"

Which Move?

Who's Ahead?

Minigo's error now can be defined as a sum of:

- **Delta between 'which move' and the moves actually explored by tree search**
- **Delta between 'who's ahead' and who actually won the game.**
- **A regularization term**

5-slide ML summary

A “**model**” is a pile of math we construct to give us answers.

We measure **error** by comparing those answers with the right answers.

We “**train**” a model by minimizing the error.

Minigo’s model outputs a “**move policy**” and a “**value estimate**”

“**Tree Search**” uses the policy and value to try to look ahead and figure out the best move in a situation.

We use tree search to play games and **create new data** to train on.

Reinforcement Learning Loop

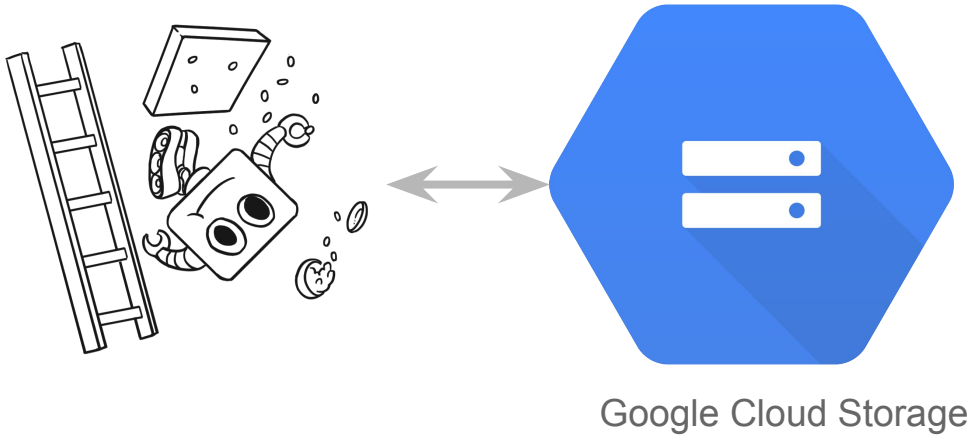
I need millions of games

1 game = x minutes, needs GPUs

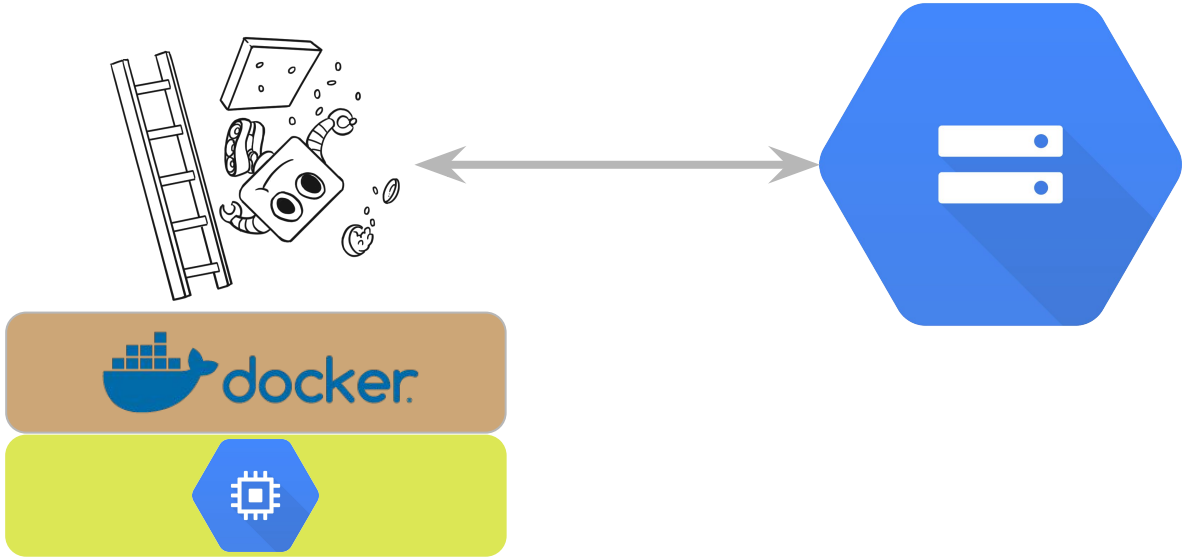
How can I scale self-play?

Kubernetes and Minigo

1. Make It Work

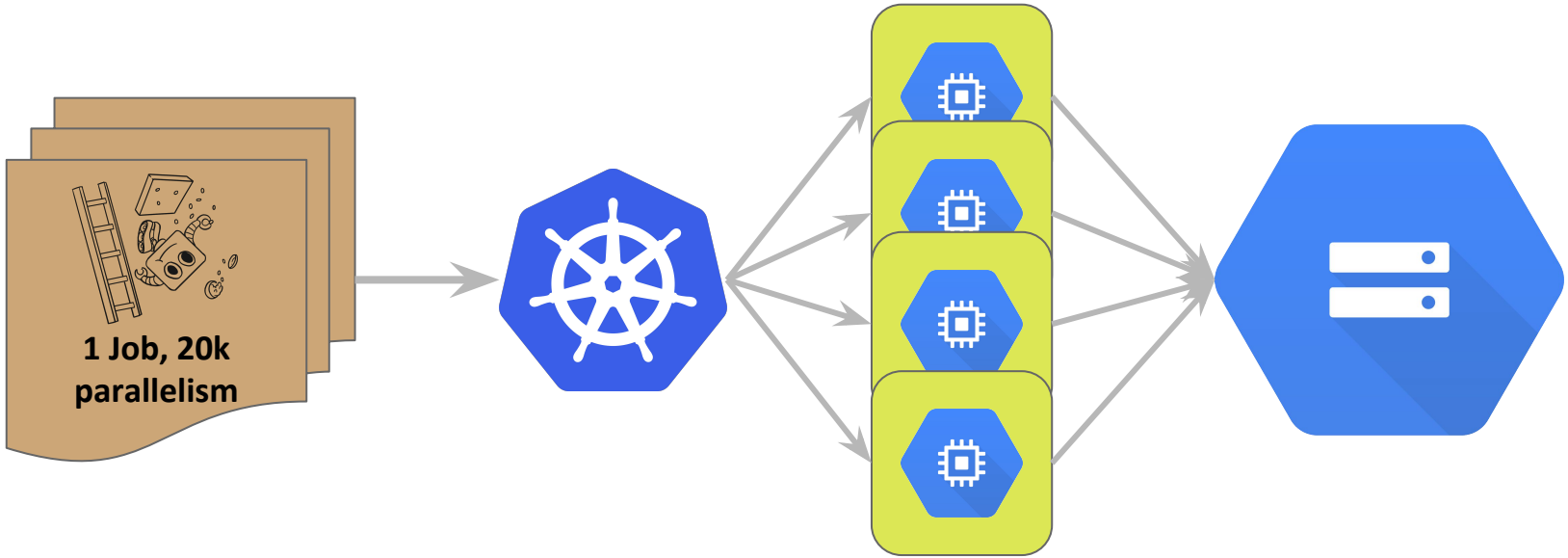


2. Make it Portable

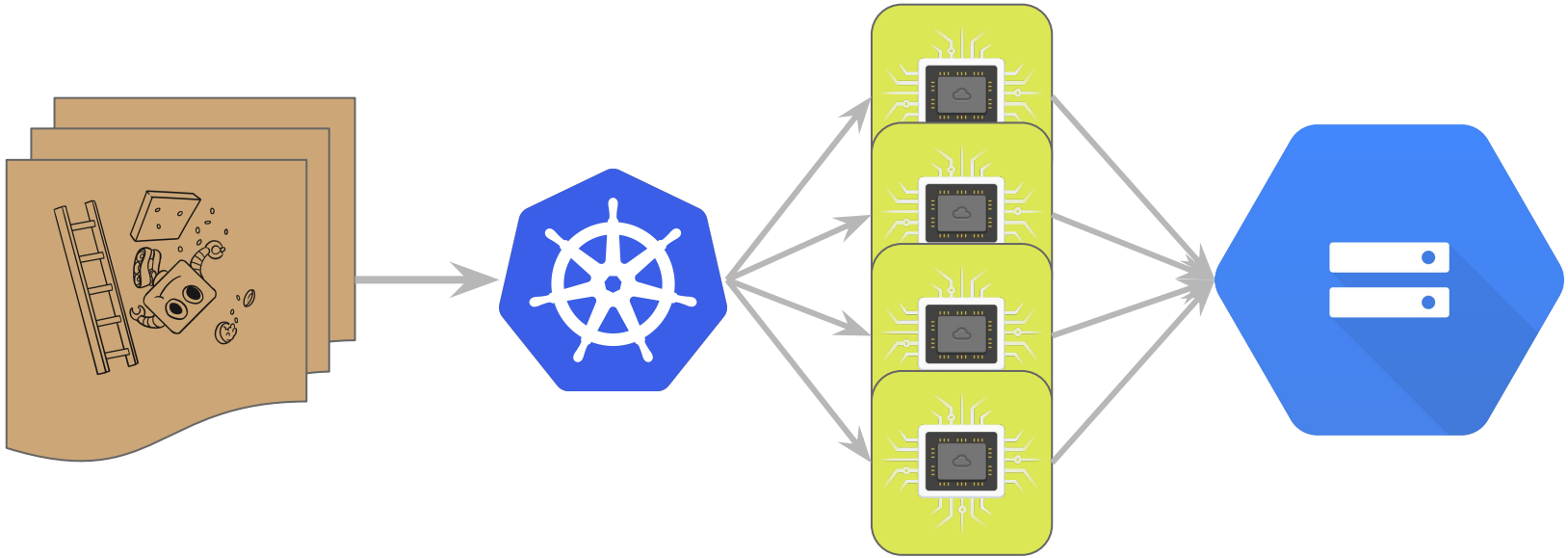


3. Make It Scale with **Kubernetes**!

3.1 Jobs as a Work Queue



4. Make it Fast with GPUs



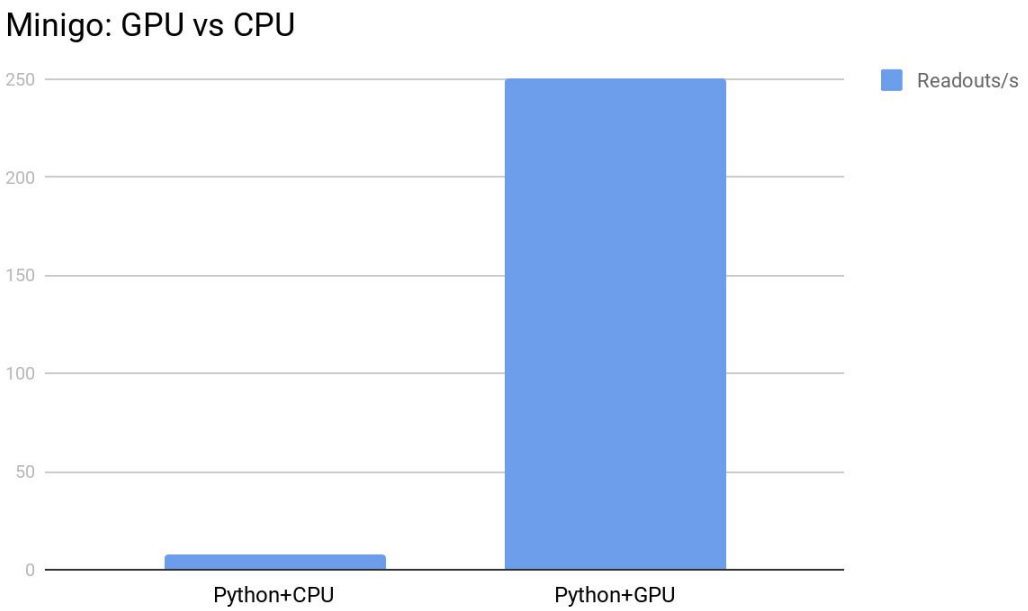
4. How Much Faster?

CPU

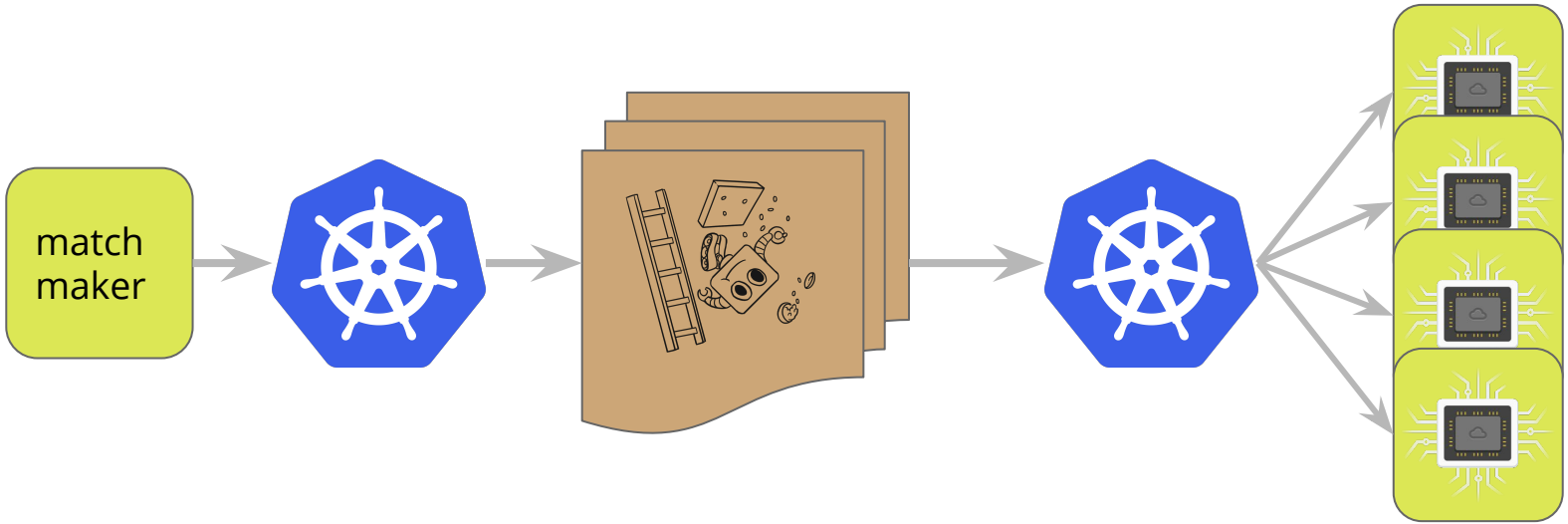
7.7 Readouts/s

GPU

250 Readouts/s



5. Evaluate Different Models



Minigo Demo

Try it Yourself

https://github.com/tensorflow/minigo/blob/master/minigui/README.md

1. Install Docker
2. Choose a Model from cloudygo.com
3. Set some environment variables. E.g.,

   ```
   export MINIGUI_BOARD_SIZE=9
   export MINIGUI_MODEL=000360-grown-teal
   ```
4. Clone Minigo: `git clone git@github.com:tensorflow/minigo.git`
5. Run: `/path/to/minigo/cluster/minigui/run-local.sh`

Results

Data: Publicly Available on GCS

cloudygo.com - Graphs, Games, Data

GCS: gs://minigo-pub/

- v5-19x19/
  - sgfs/
  - models/
- v3-9x9/
  - sgfs/
  - models/

Future Work

- TPUs
- Integration with Kubeflow
- Argo

Questions?