WORLD CLASS AI

info@adagrad.ai
+1 (678) 505-1485 | +91 8888-668811

Purpose

This technical challenge is for candidates to demonstrate their problem solving and learning abilities. All tasks in this challenge are based on real-life examples of what you could work on at Adagrad AI. The deadline for the challenge is **seven** days after you receive the document. You may submit before the deadline if you wish so. We hope that you not only solve the challenge but learn something new!

Rules- Read this first!

1. Don't be intimidated by the long document! The reason it's long is because we've tried to get as many details as possible so it's easier to understand.
2. You'd have to use PyTorch for this challenge (~~TensorFlow sucks~~ we prefer PyTorch over TensorFlow since we heavily use it to train models)
3. GPU would be required for the bonus section of the challenge. It is recommended that you sign up for Google Cloud, AWS, or Microsoft Azure for a trial if you don't have a GPU locally. The sign up will give you $200 in free credits which is more than enough to solve the assessment. **Please do not mail us to provide GPUs for you, we'd ignore e-mails like these.**
4. You may choose to refer someone else's code by giving proper credits (e.g.: including it in the docstring) In fact, you're going to have to sift through a lot of API documentation and GitHub samples to solve the challenge.
5. You should **keep your code simple and focus on the readability of your code**. Include any instructions for running and reading your code in a README file. We value thoughtfully written clean, and communicative code so other contributors can easily understand and build on top of it. Use proper variable names for functions, variables, and classes. All functions used should have a docstring. **If you fail to comply with the instructions, your submission will be summarily rejected**. FYI: We use Google-style for Python[1] and C++[2].
6. Since you're going to submit via email, it is advised to refrain from uploading it on GitHub. If you have to do that, make sure the repository is set to private[3].
7. If you've finished the challenge ahead of time, you are advised to continue with the bonus section for extra brownie points.
8. Your priorities (in order) should be:
    a. Successfully solving the assessment.
    b. Making the solution readable.
    c. Create properly labelled charts/graphs to demonstrate the results.
    d. Writing a short essay on your findings.
    e. Proper memory management.
    f. Solving the bonus section.
9. There are **no bonus points** for solving the assessment before the deadline. A well-thought solution is better than a hastily submitted one.
10. Once you're done with the assessment challenge, head over to the bonus section. We usually receive large number of applicants (last time the number was 1540) so the bonus section is where you could stand apart from the competition!
11. Learn something new! You are expected to learn something new after you complete the challenge. This is not going to be an easy challenge, so once you solve it make sure you share it with your friends on LinkedIn, Twitter, and so on!

info@adagrad.ai
+1 (678) 505-1485 | +91 8888-668811

12. You'll be given appropriate feedback: we believe feedback is pivotal for a successful career. By choosing to invest your valuable time in solving the hiring challenge, we'll guarantee you'll receive very detailed feedback on your implementation and learn something new. There might be some delay (rarely) but you can expect about 2 weeks. If you haven't heard from us in over 3 weeks, please don't hesitate to remind us (and please refrain from doing it before that!)

The Challenge

TL;DR: Use a pre-trained ResNet-18 model and implement a custom reduction operator in ONNX C++ to perform image similarity on the given dataset

Adagrad is an computer vision research company, meaning, you'll be extensively working on putting cutting-edge research into production. In doing so, most of the times, we ought to export models into a framework-neutral ONNX format through "operators" (so we can put the models to production on clouds). All the well-known operations in the Neural Networks world have corresponding ONNX operators, for example, `torch.nn.Linear` would have an operator which represents a multi-layer perceptron, `torch.nn.Conv2d` would be for the convolution operation and so forth. But what happens if a new state-of-the-art models comes up with an entirely novel architecture (something like Deformable Convolutions)? How would one go about exporting such models into the ONNX format? The answer is, custom ONNX operators!

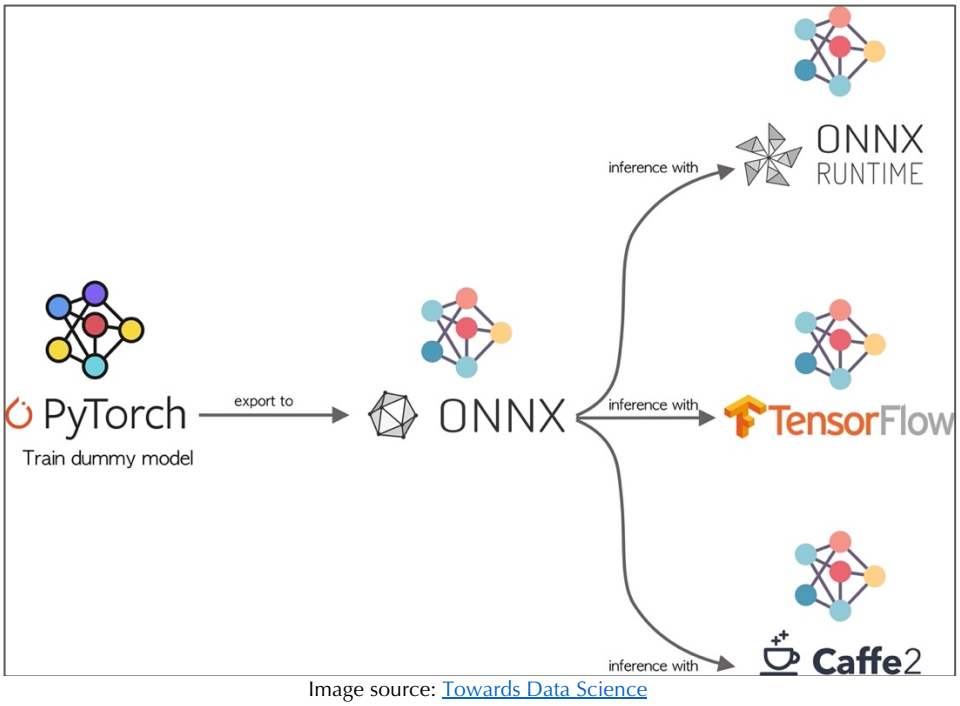


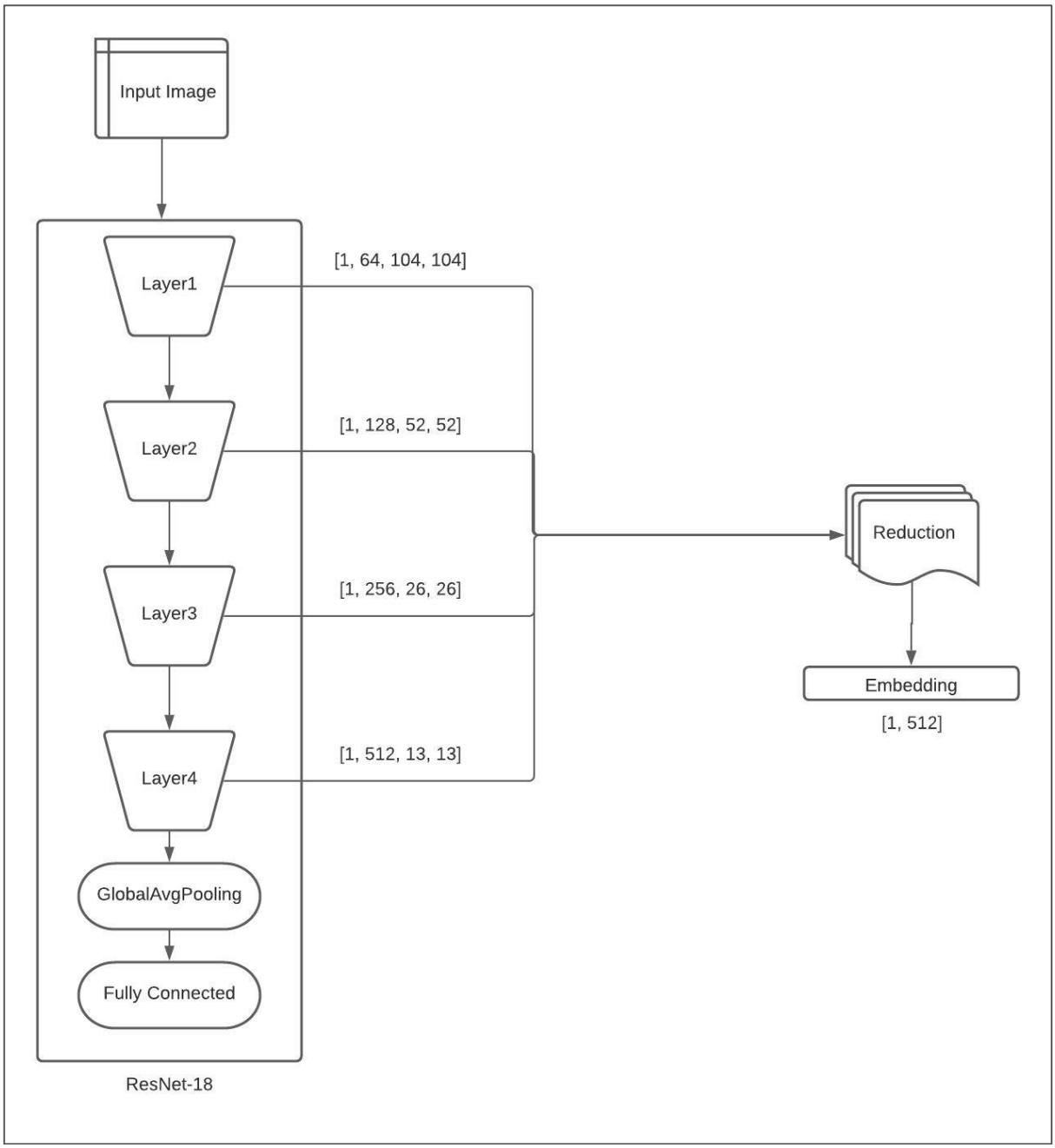
Image source: Towards Data Science

Why use C++ with ONNX? Well, not that Python is slower than C++, you'd end up calling the same C++ APIs when using Python front end. But, the raison d'être for the C++ front end is **latency**. Imagine building a self-driving car model that takes half a second to respond to a pedestrian on the road. The results would be disastrous[4]. Furthermore, we can't perform

info@adagrad.ai
+1 (678) 505-1485 | +91 8888-668811

hardware-level optimizations using Python which we would otherwise do while using C++ like strong memory management, using CPU and GPU cores efficiently (multithreading on Python is a joke), compiler level optimizations, and so on.

We'll create a novel-model which we'll call **ReductionResNet**. First, let's start with a **pretrained** ResNet-18 model from `torchvision` which has 4 layers (4 layers with a pair of `ResBlock` each). Now we would like to extract features from each of the layer and use that as a features (rather than only using the output of the last layer) Following is a picture of something we'd like to do.



Reduction operator architecture diagram using ResNet-18

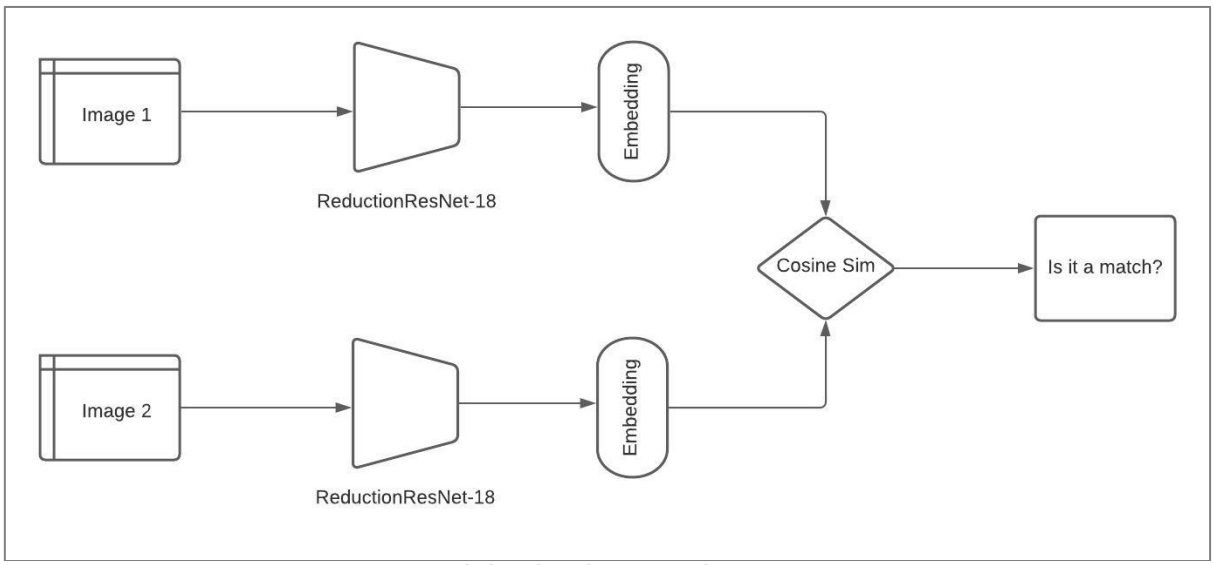
info@adagrad.ai
+1 (678) 505-1485 | +91 8888-668811

In the above diagram, consider an input image of resolution 416x416 the diagram indicates the expected output dimensions of the 4 layers respectively. In the output dimensions, for instance for the first layer, the output `[1, 64, 104, 104]` indicates `[B, C, H, W]` where B, C, H, W are the batch size, channel size, height, and width of the output maps. The height and width would change if the input resolution is different, but the number of channels would remain constant.

Note

Do not confuse output maps with the weight of the convolution kernels. Weights are constant at inference time but output maps change with every image!

Next, we would like to perform a **reduction** operation on the output of the kernel maps and produce an embedding of size [1, 512] this would be nothing but a 512-size array (embedding) that's the representation of the input image. We would then use cosine similarity to compare the distance between two images, and if the similarity is greater than **0.75** we would say that the image are **similar** images. Use the same images that are provided in the challenge and report the similarity in your report.



High-level architecture diagram

Reduction Operator (the C++ part)

The reduction operator takes input from the 4 layers of ResNet, performs a repeat-interleave operation to make the dimensions equal and then takes the average of all the 4 arrays to output a [1, 512] dimension embedding. The task is to write two functions, first which is the reduction operator and second, which is a helper function which performs the repeat-interleave operation.

info@adagrad.ai
+1 (678) 505-1485 | +91 8888-668811

You may use `torch.nn.AdaptiveAvgPooling` to perform average pooling on the 4D tensors to convert it to a 1D tensor. You ought to implement two functions in your `src` folder with the following signature:

```
/// Implements the Reduction ONNX operator by interleaving the
/// \param layerOne:   layer 1 output after AdaptiveAvgPooling [1x64] or [64]
/// \param layerTwo:   layer 2 output after AdaptiveAvgPooling [1x128] or [128]
/// \param layerThree: layer 3 output after AdaptiveAvgPooling [1x256] or [256]
/// \param layerFour:  layer 4 output after AdaptiveAvgPooling [1x512] or [512]
/// \return torch::Tensor: reduction output of the tensors [1x512] or [512]
torch::Tensor reduction (
    torch::Tensor layerOne,
    torch::Tensor layerTwo,
    torch::Tensor layerThree,
    torch::Tensor layerFour) {
  ...
}
```

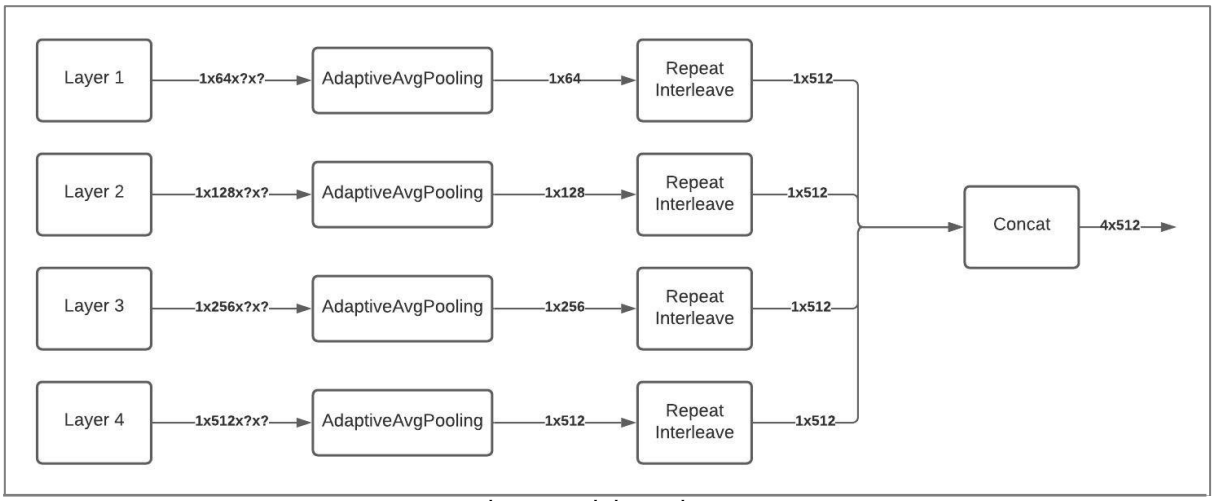
And repeat interleave:

```
/// Implements the repeat interleave operation
/// \param input: torch 1-dimensional Tensor
/// \return torch::Tensor: interleaved 1-dimensional result
torch::Tensor repeatInterleave(torch::Tensor input) {

}
```

Repeat Interleave (C++)

You can read about repeat-interleave in the PyTorch docs[5]. The reason for using this operation is to expand the dimensions of the early layers (layer 1, 2, 3) to match that of the layer 4 (512) so that you can perform simple average on concatenated [4, 512] tensors.



Repeat-interleave and the reduction operator

info@adagrad.ai
+1 (678) 505-1485 | +91 8888-668811

ONNX Integration (C++ / hacking)

In order to use the custom ONNX operator, you need to write a small wrapper in C++. The steps involve using `setuptools`, and `torch.utils.cpp_extension` to build an object file that you can import using `torch.ops.load_library`.

Detailed steps can be followed from the official tutorial[6] which does something almost similar to what we're trying to do. This is one of the most important piece of documents you will go through, so read it carefully!

Getting Everything together (Python!)

Once you've loaded your custom operator, next you gotto implement simple image similarity algorithm using cosine similarity. You may use any library you wish e.g.: `torch.nn.CosineSimilarity` from PyTorch or pairwise cosine similarity from scikit-learn `sklearn.metrics.pairwise.cosine_similarity`. Now, for every given image in the dataset, calculate the number of dissimilar (unique) images as the end result.

Bonus Section (CUDA C/C++)

The reduction operator would perform operations on the CPU. Operations like mean-reduction are embarrassingly parallel[7] meaning, they can be highly optimized on a GPU that usually has thousands of cores. To do this, you'd have to write a CUDA kernel which would launch the mean reduction operations on the GPU. A sample tutorial can be found in the PyTorch documentation[8]. The compiling process is exactly same as before.

You can further optimize your inference pipeline using proper C++ memory management. Following are some tips:

1. Using smart pointers to automate memory management.
2. Using move semantics to reduce overhead and unnecessary copying.
3. Using tools like valgrind, cuda-memcheck, heaptrack to profile your memory usage and find memory leaks.

Some helpful tips, and words of wisdom..

1. Premature optimization is the root of all evil – Donald Knuth. Don't try to write CUDA kernels if you don't understand how it works. Maybe start with the simple CPU based implementation before you head over to GPU programming.
2. Search on GitHub! 97.33% of the problems in the world are already solved and are available on GitHub. Don't reinvent the wheel. Take it as a reference and develop your own solution. Don't just mindlessly copy-paste, we'd know if you do so (~~experience?~~). At least understand how it works lol.

info@adagrad.ai
+1 (678) 505-1485 | +91 8888-668811

3. 99.99% of things you'd need to solve the assessment are present in the documentation. Please read them!

4. Ask a lot of questions via email (mentioned below) if stuck or if you're unclear on certain things. Please refrain from sending multiple emails by collating all your questions in a single mail since the research team is extremely busy these days.

Deliverables

Submit the following files:

1. Folder **research**: will have the experimentation/scratch code in Python. The files that you came up with when performing inference on your model which can be pythons scripts or Jupyter notebooks.
2. Folder **inference**:
    a. **src**: C++ code for inference with the ONNX operator.
    b. **CMakeLists**: the CMake file that will ensure your build compiles on other devices.
    c. **lib**: all the required dependencies and the lib generated for the ONNX operator. You may choose to link dynamically, in that case, you can specify that in the **README** file. Use static linking wherever possible.
    d. **includes** all the header files required by your solution.
3. Folder **models:** This folder should contain your resulting ONNX file which can be viewed using Netron. You can save your PyTorch model to ONNX using `torch.onnx.export`. You need to figure out how to export custom operators ☺
4. Folder **findings:** will have the docx reports, charts, or anything that you have observed and understood through the experiment. Imagine we're trying to publish a paper in NeurIPS, or CVPR.
5. **README**: listing the instructions that are required for the setup. Additionally, your README should contain your personal info like name, phone number, current job, and so on.
6. Setup **scripts**, helper scripts, other files: if you think that will help us speed up evaluating your solutions.
7. **Your story document**. Everyone has a story, this document will help us know you more on a personal level. In this section, write a **short essay** in a Word document about

info@adagrad.ai
+1 (678) 505-1485 | +91 8888-668811

yourself and what shapes you as a human. What are your ideals, and philosophies? Some questions that you should definitely answer are included below. In the questions below you might be presented with two choices (e.g.: is the name of university important, or the course you study) you are allowed to only **choose one**. Please be **clear, and unequivocal** in your response and avoid generic or sweeping statements.

> Note
> *The story document is one of the most critical parts of the assessment as it helps us to know you. We open it even before reading a single line of code. Submissions without this document will be* ***summarily rejected!***

Questions:

a. Why should I ditch someone who's more qualified and has a better solution than you and hire you instead?
b. What's more important- the institute name or the course you study?
c. What was the biggest setback in your life, and how did it change you as a person?
d. List at least 3 things you've learned about life?
e. What would you do differently if nobody judged you?
f. When do you work at your best? Individually, or in a team?
g. What's more important as a leader in both positive and negative outcomes? Taking the responsibility for the team, or holding each other's accountable?
h. What do you think about work-life balance?

That's it! Put all the above documents in a folder called FirstName and zip the folder and call it yourname_company.zip or yourname_college.zip. For instance, if my name is John Doe and I'm from Adagrad, my folder will be called JohnDoe and will be zipped in john_adagrad.zip. Email this to niranjan@adagrad.ai with subject as "Machine Learning Engineer 2021: <your full name here>" If you're uploading the ZIP on Google Drive make sure you're providing access to **Anyone over internet**. Double check my opening the link in an incognito window before submitting.

References

[1] Google Python Style-guide: http://google.github.io/styleguide/pyguide.html
[2] Google C++ Style-guide: https://google.github.io/styleguide/cppguide.html
[3] Setting GitHub repo to private: https://help.github.com/en/github/administering-a-repository/setting-repository-visibility#making-a-repository-private
[4] Death of Elaine Herzberg: https://en.wikipedia.org/wiki/Death_of_Elaine_Herzberg
[5] PyTorch Repeat-Interleave: https://pytorch.org/docs/stable/generated/torch.repeat_interleave.html
[6] ONNX Operator Tutorial: https://github.com/onnx/tutorials/blob/master/PyTorchCustomOperator/README.md
[7] Embarrassingly Parallel: https://en.wikipedia.org/wiki/Embarrassingly_parallel
[8] PyTorch GPU Operator Tutorial: https://brsoff.github.io/tutorials/advanced/cpp_extension.html

Goodspeed! – Adagrad AI Team