

# **Lightweight, low-overhead, high-performance: machine learning directly in C++**

Ryan R. Curtin  
Atlanta, GA  
[ryan@ratml.org](mailto:ryan@ratml.org)

November 2, 2023

# Geographical Origin: *Atlanta*



# Geographical Origin: Atlanta



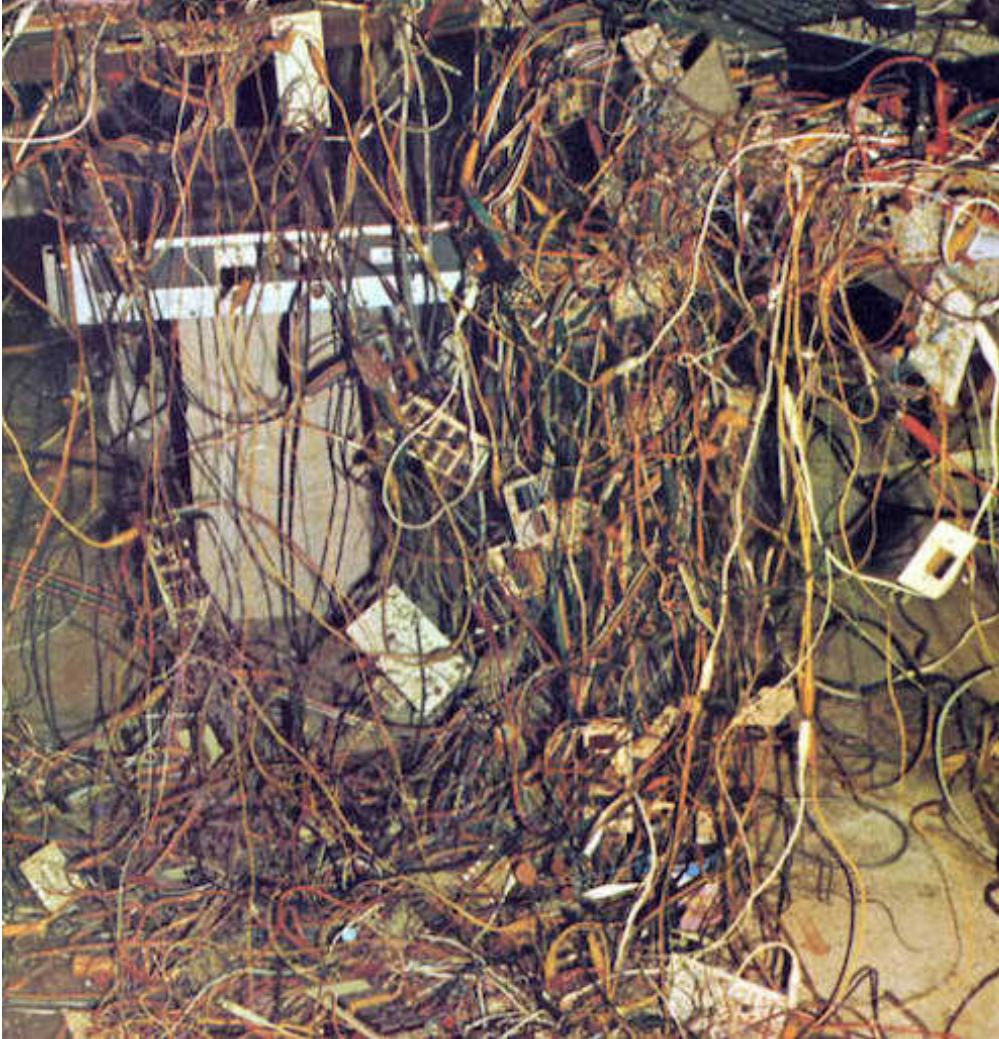
# Geographical Origin: Atlanta



## Atlanta quick tips:

- Use the Plane Train unless traveling between concourses A and B
- Be hungry
- Do not go to the Varsity
- Check out the Laser Light Show at Stone Mountain and bring lots of cheap beer

# The Problem



*Typical machine learning deployment*

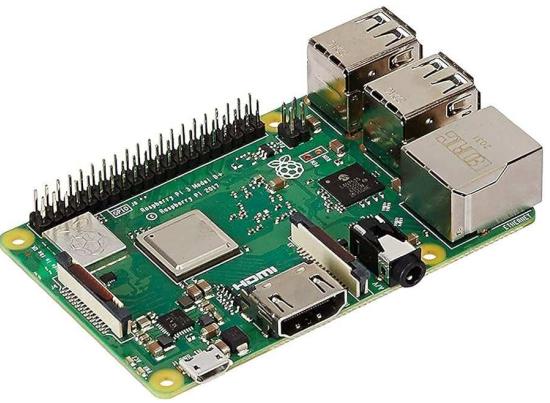
# The Problem



*Typical machine learning deployment*

~60 packages needed for a basic miniconda scikit-learn or TensorFlow environment...

# Size Is Not Always Cheap



# **What is machine learning?**

1. A series of linear algebra operations or other mathematical operations.

# **What is machine learning?**

1. Turn the data into numbers.
2. A series of linear algebra operations or other mathematical operations.

# **What is machine learning?**

1. Turn the data into numbers.
2. A series of linear algebra operations or other mathematical operations.
3. Get the output into the right format.

# What is machine learning?

1. Turn the data into numbers.
2. A series of linear algebra operations or other mathematical operations.  
**Typically done in FORTRAN, C, or C++!** (BLAS, LAPACK, OpenBLAS, etc.)
3. Get the output into the right format.

# What is machine learning?

1. Turn the data into numbers.
2. A series of linear algebra operations or other mathematical operations.  
**Typically done in FORTRAN, C, or C++!** (BLAS, LAPACK, OpenBLAS, etc.)
3. Get the output into the right format.



# A Collection Of Hellos World

C++:

```
#include <cstdio>  
  
int main() { printf("Hello world!\n"); }
```

Python/R:

```
print("Hello world!")
```

Julia:

```
print("Hello world!\n")
```

Scala:

```
object hello {  
  def main(args: Array[String]) = {  
    println("Hello world!")  
  }  
}
```

# How big is your Hello World?

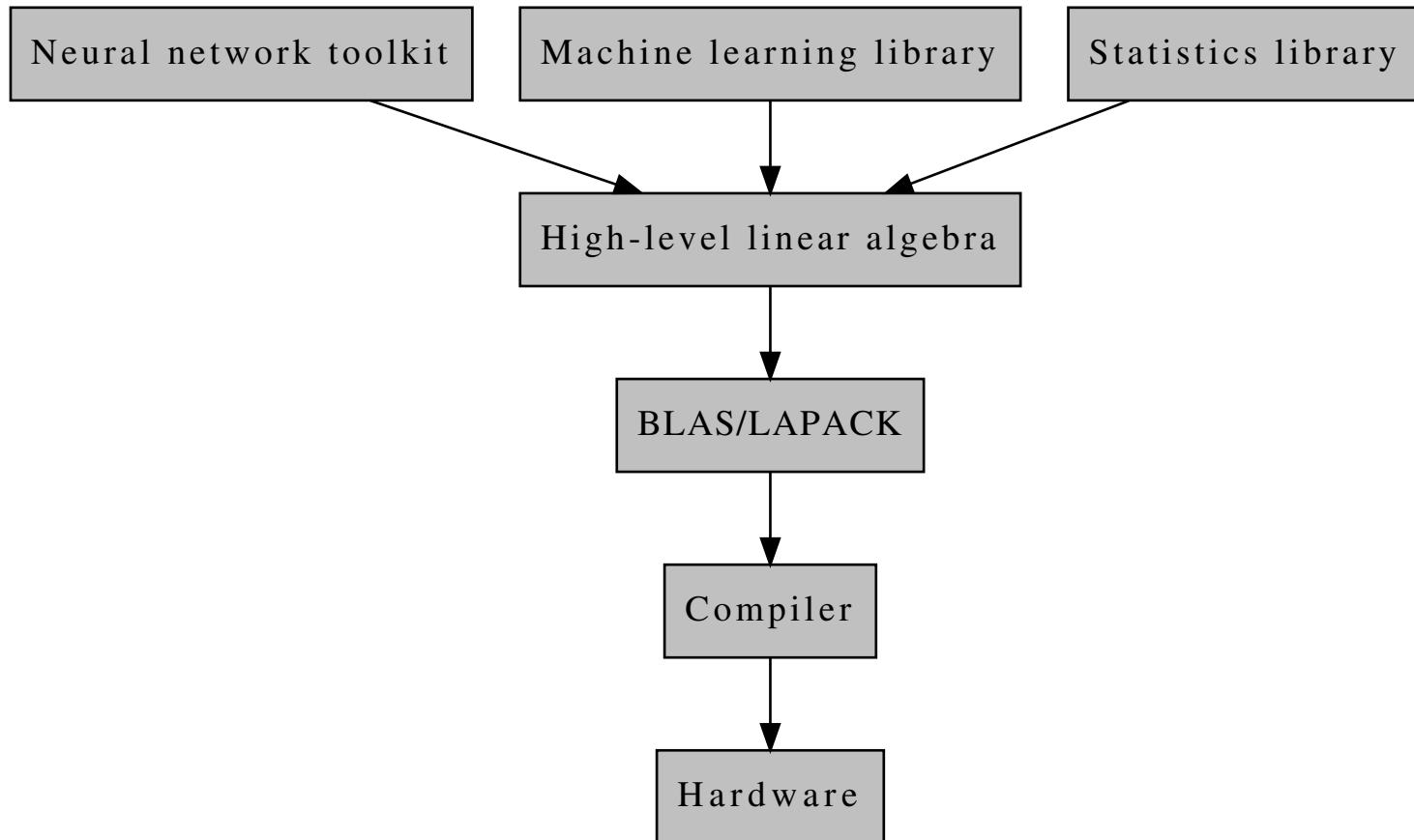
Containers only contain hello world script and minimal environment needed to run.

\$ docker images	REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
	hello_world_r	latest	ee64478b9235	35 seconds ago	811MB
	hello_world_scala	latest	53fc54fa84a9	3 minutes ago	751MB
	hello_world_jl_alpine	latest	15aecf1e9146	14 minutes ago	535MB
	hello_world_jl_full	latest	eaca21a0a0e0	14 minutes ago	606MB
	hello_world_py_full	latest	090088f4972d	16 minutes ago	1.02GB
	hello_world_py_slim	latest	cfdbf9ffce45	17 minutes ago	130MB
	hello_world_py_alpine	latest	a0ffd6653f17	18 minutes ago	51.8MB
	hello_world_cpp	latest	6dbe01398789	22 minutes ago	7.71MB
	hello_world_cpp_distroless	latest	32ae3c33e2f2	25 minutes ago	2.7MB

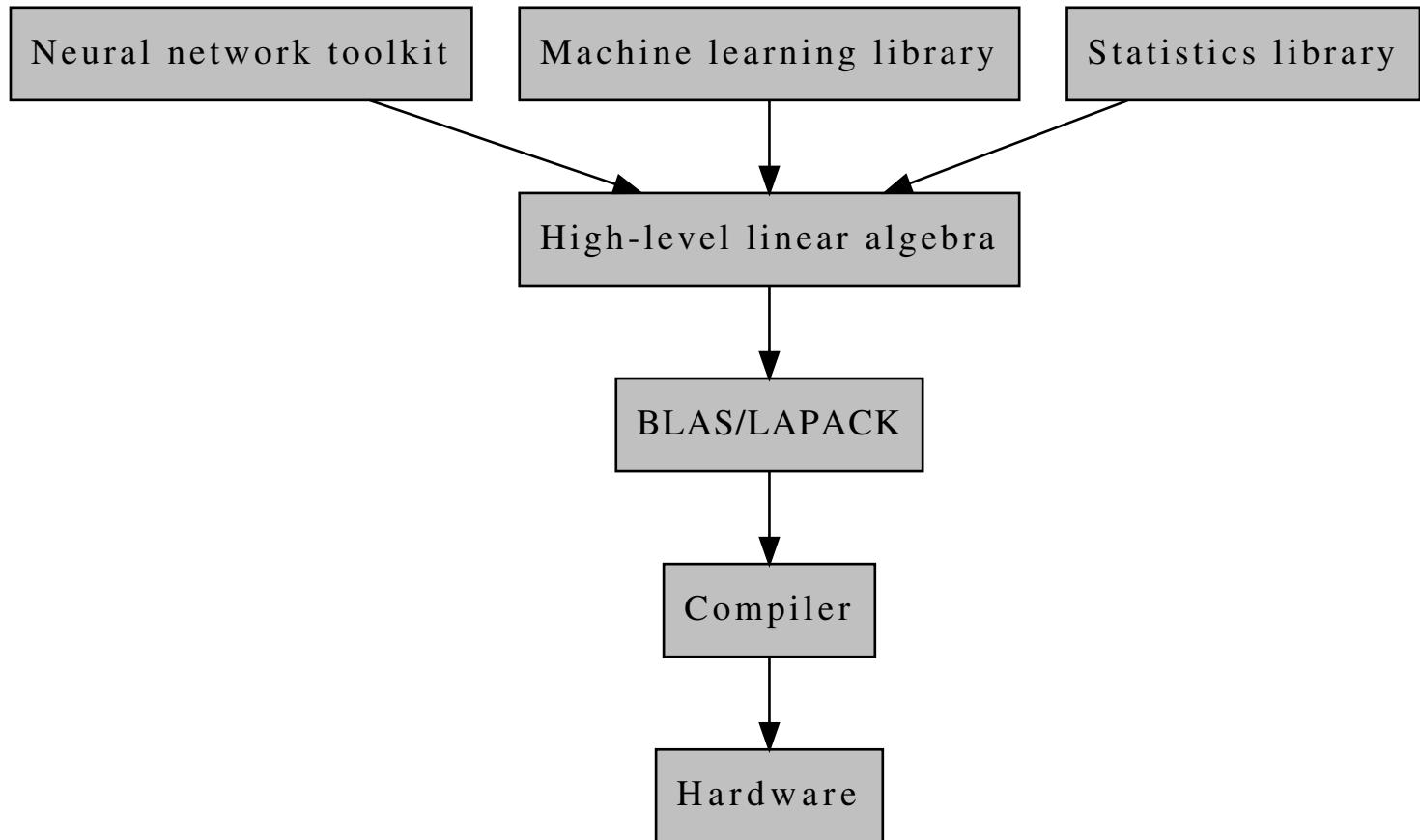
Built using base containers: r-base, openjdk:22-slim, julia:1, julia:1-alpine, python:3-full, python:3-slim, python:3-alpine, alpine, distroless/static-debian11.

More size optimization possible for C++, I didn't try too hard...

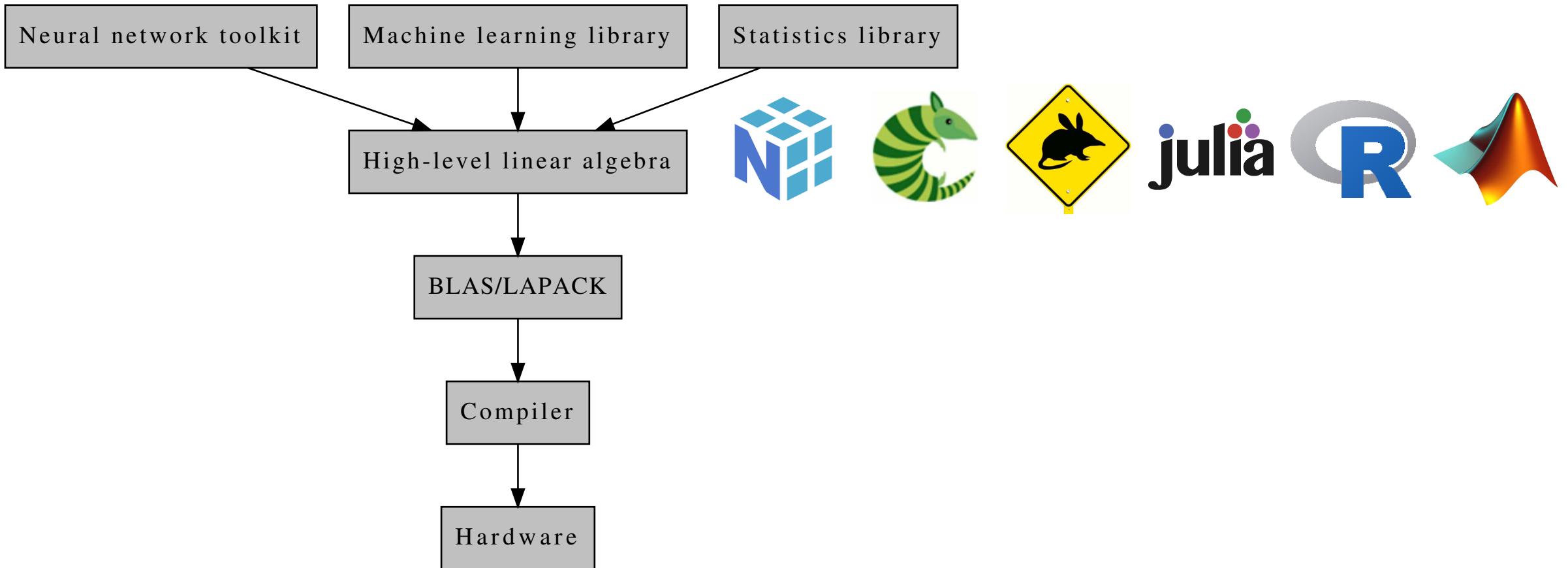
# The machine learning stack



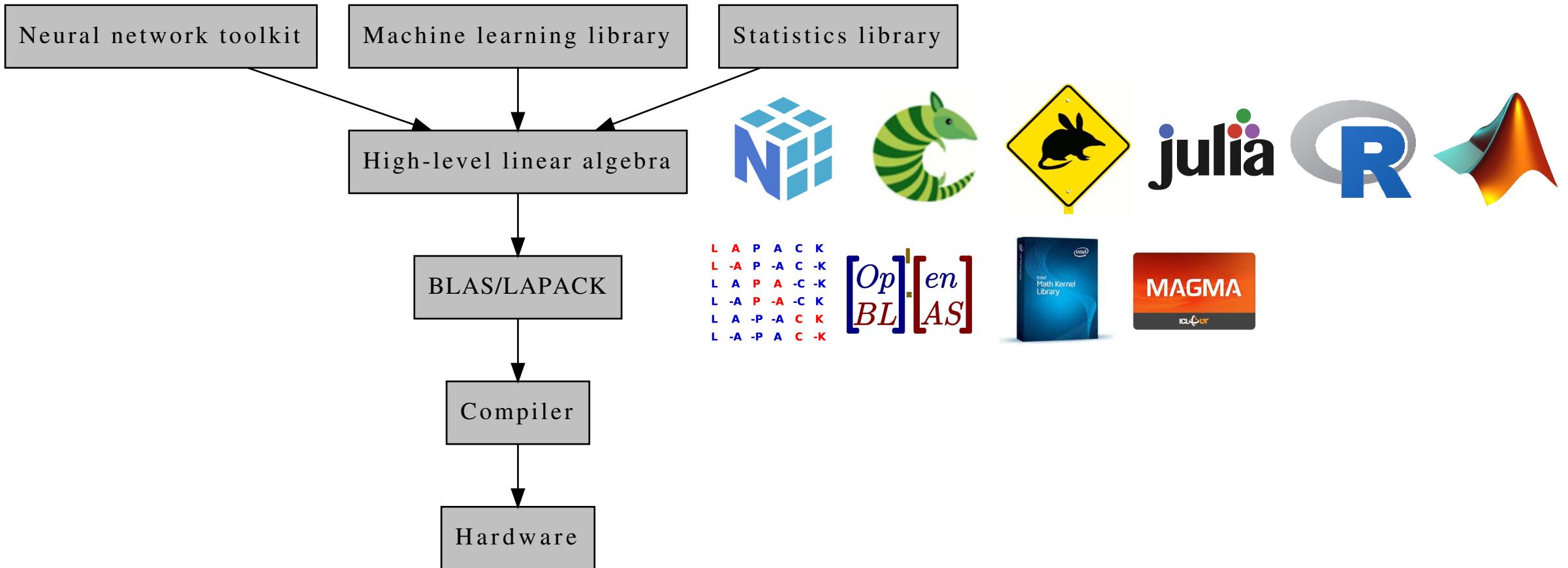
# The machine learning stack



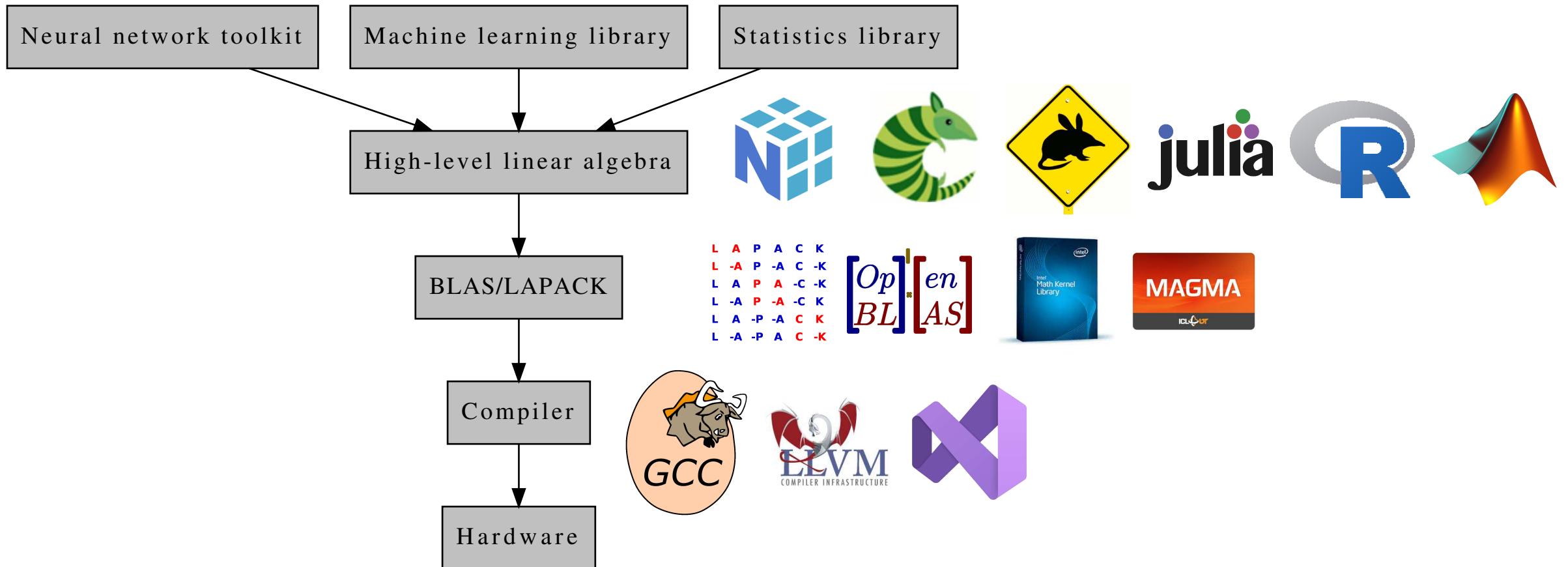
# The machine learning stack



# The machine learning stack



# The machine learning stack



# **C++ is great!**

Pros of C++:

- No runtime other than what the system already has

# C++ is great!

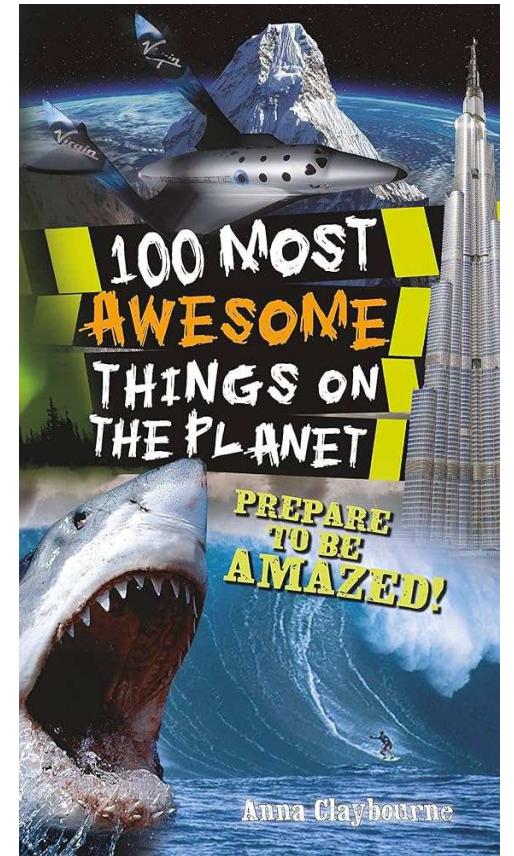
Pros of C++:

- No runtime other than what the system already has
- Very easy and light interfaces with C and FORTRAN

# C++ is great!

Pros of C++:

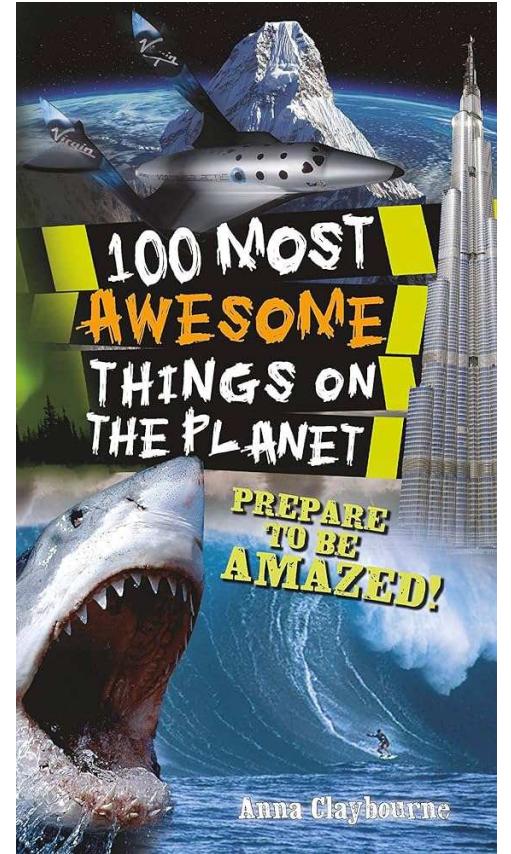
- No runtime other than what the system already has
- Very easy and light interfaces with C and FORTRAN
- Generic programming with templates for compile-time code generation



# C++ is great!

Pros of C++:

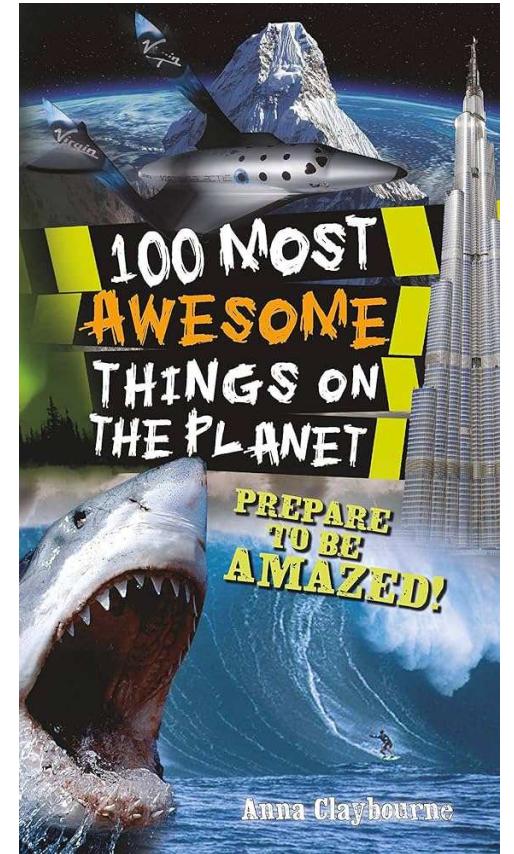
- No runtime other than what the system already has
- Very easy and light interfaces with C and FORTRAN
- Generic programming with templates for compile-time code generation
- Low-level memory management for fine-grained control



# C++ is great!

Pros of C++:

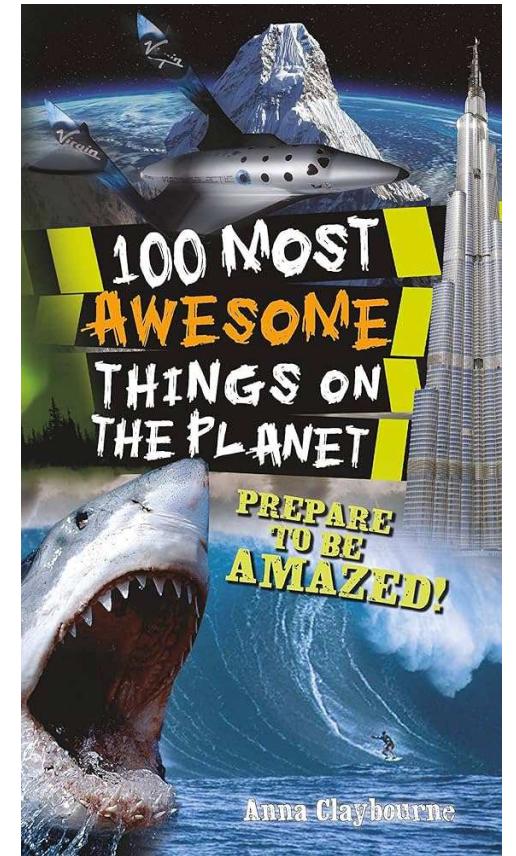
- No runtime other than what the system already has
- Very easy and light interfaces with C and FORTRAN
- Generic programming with templates for compile-time code generation
- Low-level memory management for fine-grained control
- Readily runs on all reasonable modern systems



# C++ is great!

Pros of C++:

- No runtime other than what the system already has
- Very easy and light interfaces with C and FORTRAN
- Generic programming with templates for compile-time code generation
- Low-level memory management for fine-grained control
- Readily runs on all reasonable modern systems
- Fast! Small!

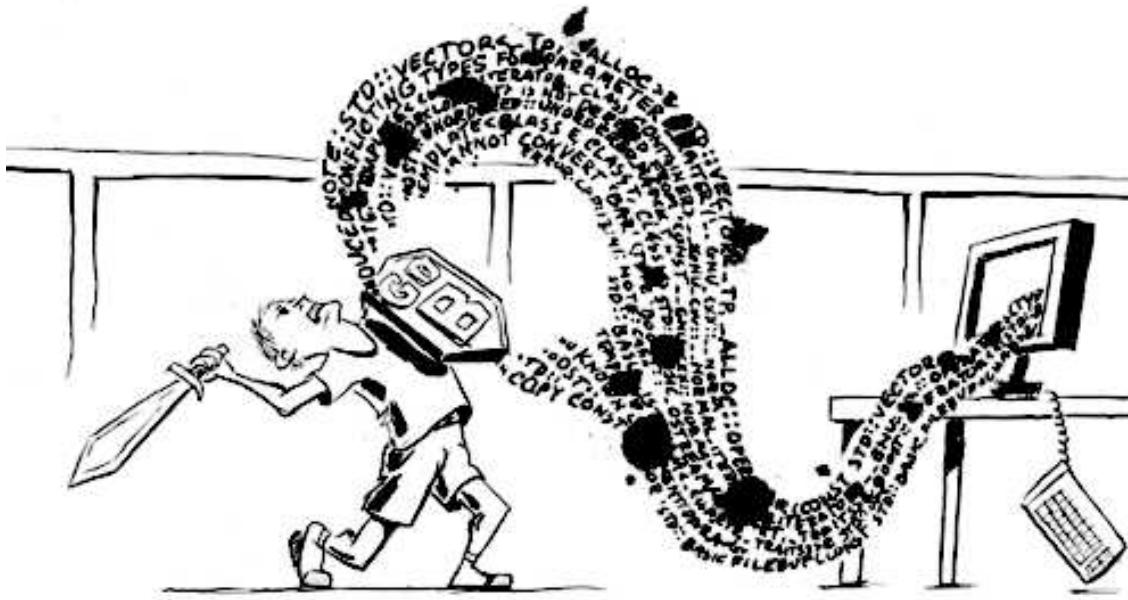


# **C++ is terrible!**

Cons of C++:

# C++ is terrible!

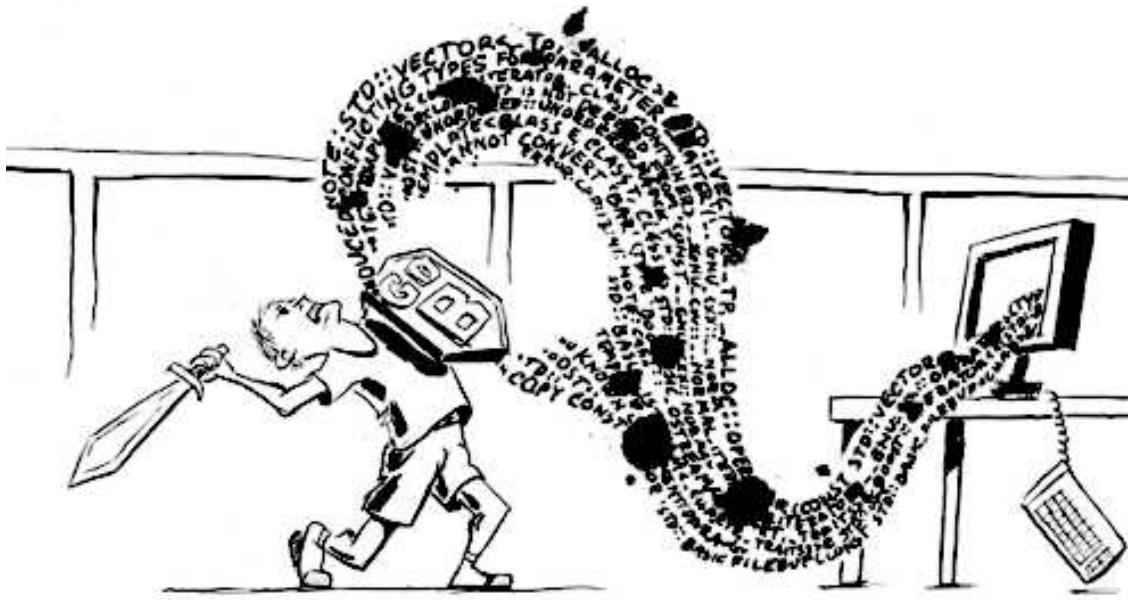
## Cons of C++:



- Templates can be hard to debug because of error messages.

# C++ is terrible!

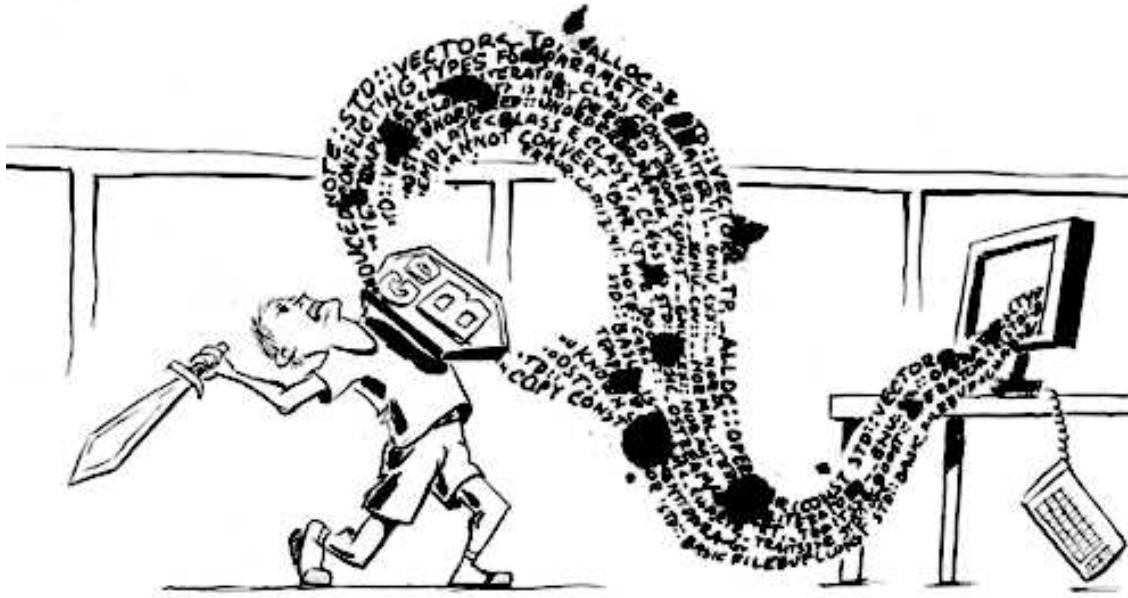
## Cons of C++:



- Templates can be hard to debug because of error messages.
  - Memory bugs are easy to introduce.

# C++ is terrible!

Cons of C++:



- Templates can be hard to debug because of error messages.
- Memory bugs are easy to introduce.
- The new language revisions are not making the language any simpler...

# C++ is terrible!

Cons of C++:



- Templates can be hard to de
  - Memory bugs are easy to in
  - The new language revisions
- impler...

# The C++ data science ecosystem



- **Linear algebra:** Armadillo, Eigen, Bandicoot (GPU), XTensor, std::mdspan, GGML...

# The C++ data science ecosystem



- **Linear algebra:** Armadillo, Eigen, Bandicoot (GPU), XTensor, std::mdspan, GGML...

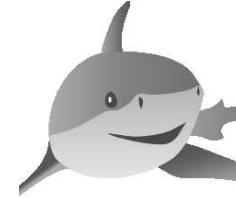
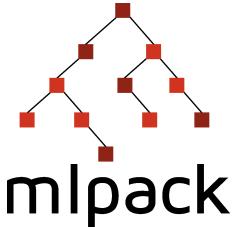
The Armadillo library gives us good linear algebra primitives:

```
using namespace arma;
mat x(100, 100, fill::randu);
mat y(100, 100, fill::randn);
mat z = (x + y) * chol(x) + 3 * chol(y.t());
```

# The C++ data science ecosystem

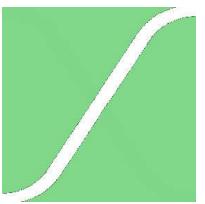


- **Linear algebra:** Armadillo, Eigen, Bandicoot (GPU), XTensor, std::mdspan, GGML...

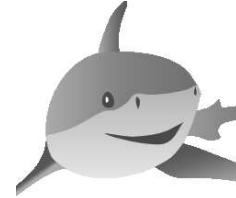
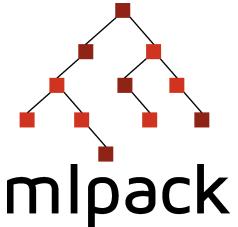


- **Machine learning:** mlpack, ensmallen, dlib-ml, PyTorch, XGBoost, LIBLINEAR, Shark, Shogun, darknet, llama.cpp, Vowpal Wabbit, TensorFlow, OpenCV, Catboost...

# The C++ data science ecosystem



- **Linear algebra:** Armadillo, Eigen, Bandicoot (GPU), XTensor, std::mdspan, GGML...



- **Machine learning:** mlpack, ensmallen, dlib-ml, PyTorch, XGBoost, LIBLINEAR, Shark, Shogun, darknet, llama.cpp, Vowpal Wabbit, TensorFlow, OpenCV, Catboost...
- **More:** xeus-cling, Jupyter, xplot, xframe, support libraries for whatever else...

# Quick Linear Algebra Benchmark #1

Task 1:  $z = 2(x' + y) + 2(x + y')$ .

```
extern int n;  
mat x(n, n, fill::randu);  
mat y(n, n, fill::randu);  
mat z = 2 * (x.t() + y) + 2 * (x + y.t()); // only time this line
```

$n$	arma	numpy	octave	R	Julia
1000	0.029s	0.040s	0.036s	0.052s	<b>0.027s</b>
3000	0.047s	0.432s	0.376s	0.344s	<b>0.041s</b>
10000	<b>0.968s</b>	5.948s	3.989s	4.952s	3.683s
30000	<b>19.167s</b>	62.748s	41.356s	<i>fail</i>	36.730s

# Quick Linear Algebra Benchmark #2

Task 2:  $z = a'(\text{diag}(b)^{-1})c$ .

```
extern int n;  
vec a(n, fill::randu);  
vec b(n, fill::randu);  
vec c(n, fill::randu);  
double z = as_scalar(a.t() * inv(diagmat(b)) * c); // only time this line
```

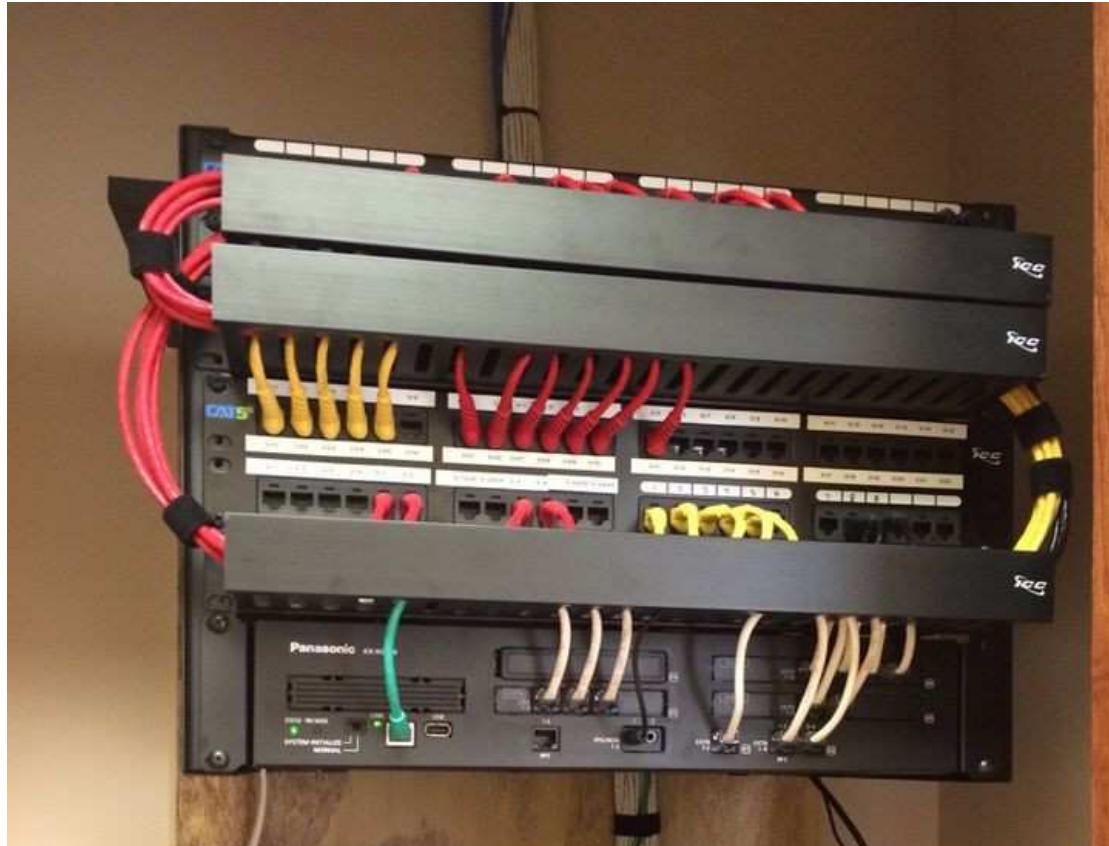
$n$	<b>arma</b>	<b>numpy</b>	<b>octave</b>	<b>R</b>	<b>Julia</b>
1k	<b>8e-6s</b>	0.100s	2e-4s	0.014s	0.057s
10k	<b>8e-5s</b>	49.399s	4e-4s	0.208s	18.189s
100k	<b>8e-4s</b>	<i>fail</i>	0.002s	<i>fail</i>	<i>fail</i>
1M	0.009s	<i>fail</i>	0.024s	<i>fail</i>	<i>fail</i>
10M	0.088s	<i>fail</i>	0.205s	<i>fail</i>	<i>fail</i>
100M	0.793s	<i>fail</i>	1.972s	<i>fail</i>	<i>fail</i>
1B	8.054s	<i>fail</i>	19.520s	<i>fail</i>	<i>fail</i>

## **Two Points (That I Hope To Convince You Of)**

- 1. It isn't ugly.** It's easy to write and read and maintain.

# Two Points (That I Hope To Convince You Of)

1. **It isn't ugly.** It's easy to write and read and maintain.
2. **The code is deployable** and you can compile it to be really small. *No bloat.*



*a more organized machine learning deployment*

# Language Detection: A Simple Model

- **Input:** user gives a string  
“Ihr Spracherkennungsmodell ist sehr schlecht.”
- **And then:** magic occurs
- **Output:** user receives a language  
“de”

# Language Detection: A Simple Model

- **Input:** user gives a string  
“Ihr Spracherkennungsmodell ist sehr schlecht.”
- **And then:**
  - Convert string into numbers
  - Use machine learning model to predict language (as a number)
  - Turn number into language
- **Output:** user receives a language  
“de”

# Language Detection: A Simple Model

- **Input:** user gives a string  
“Ihr Spracherkennungsmodell ist sehr schlecht.”
- **And then:**
  - Convert string into numbers; features:
    - 256 byte counts*
    - 768 of the most common 2-byte sequences in the training data*
    - Normalize resulting 1024-dimensional vector*
  - Use machine learning model to predict language (as a number)
  - Turn number into language
- **Output:** user receives a language  
“de”

# Language Detection: A Simple Model

- **Input:** user gives a string  
“Ihr Spracherkennungsmodell ist sehr schlecht.”
- **And then:**
  - Convert string into numbers; features:
    - 256 byte counts*
    - 768 of the most common 2-byte sequences in the training data*
    - Normalize resulting 1024-dimensional vector*
  - Use machine learning model to predict language (as a number)
    - Softmax regression (multi-class logistic regression)*
    - Number of classes = number of languages*
  - Turn number into language
- **Output:** user receives a language  
“de”

# Language Detection: A Simple Model

- **Input:** user gives a string

“Ihr Spracherkennungsmodell ist sehr schlecht.”

- **And then:**

Convert string into numbers; features:

*256 byte counts*

*768 of the most common 2-byte sequences in the training data*

*Normalize resulting 1024-dimensional vector*

Use machine learning model to predict language (as a number)

*Softmax regression (multi-class logistic regression)*

*Number of classes = number of languages*

Turn number into language

*Map class id back into name of language*

- **Output:** user receives a language

“de”

# n-gram computation code (or a simplification)

```
void ComputeNGrams(const vector<string>& stringData, mat& ngrams)
{
    // Resize the matrix to the right size.
    // This uses 256 dimensions for the single-byte features.
    // Armadillo and mlpack use column-major data, so each string is a *column*.
    ngrams.zeros(256, stringData.size());
    for (size_t i = 0; i < stringData.size(); ++i)
    {
        for (size_t j = 0; j < stringData[i].size(); ++j)
        {
            // Add one to the count of times we've seen this byte.
            ++ngrams((unsigned char) stringData[i][j], i);
        }

        // Now normalize. (Important since not all strings are the same length.)
        ngrams.col(i) /= accu(ngrams.col(i));
    }
}
```

# NGramLangDetectModel

```
class NGramLangDetectModel
{
public:
    // Train the model on the given data.
    void Train(const vector<string>& data, const vector<string>& labels,
               const size_t bigramsToKeep = 768);

    // Classify a single string, or a set of strings.
    string Classify(const string& point) const;
    void Classify(const vector<string>& data, vector<string>& predictions) const;

    // Serialize the model to/from disk.
    template<typename Archive> void serialize(Archive& ar, const unsigned int /* version */);

private:
    // The actual trained model.
    mlpack::SoftmaxRegression model;
    // The set of 2-byte sequences we are using as features.
    unordered_map<uint16_t, size_t> bigramDimensionMap;
    // The names of each class.
    vector<string> classNames;
};
```

## Train()

```
void NGramLangDetectModel::Train(const vector<string>& data, const vector<string>& labels,
                                 const size_t bigramsToKeep)
{
    // Convert data to numeric data via n-gramming.
    mat trainData;
    ComputeNGrams(data, bigramDimensionMap, trainData, bigramsToKeep);
    // Convert labels to numeric data. We also have to keep track of the class names.
    Row<size_t> trainLabels(trainData.n_cols);
    unordered_map<string, size_t> classMappings; // Stores language -> class index mappings.
    for (size_t i = 0; i < labels.size(); ++i)
    {
        // If we haven't seen this label before, assign the next index to this label.
        if (classMappings.count(labels[i]) == 0)
        {
            classMappings[labels[i]] = classMappings.size();
            classNames.push_back(labels[i]); // Add the label to the list of class labels.
        }
        trainLabels[i] = classMappings[labels[i]];
    }

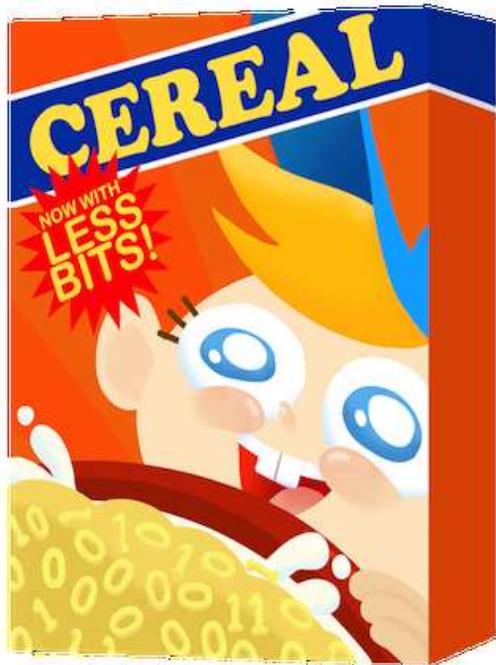
    // Train the model. It would be possible to set some hyperparameters here.
    model = mlpack::SoftmaxRegression(trainData, labels, classNames.size());
}
```

## Classify()

```
// Predict the class of one string.  
string NGramLangDetectModel::Classify(const string& point) const  
{  
    vec processedPoint;  
    ComputeNGrams(point, processedPoint, bigramDimensionMap); // Turn string into numeric features.  
    const size_t classIndex = model.Classify(processedPoint); // Use model to classify.  
    return classNames[classIndex];  
}  
  
// Predict the class of a set of strings.  
void NGramLangDetectModel::Classify(const vector<string>& data, vector<string>& predictions) const  
{  
    mat points;  
    ComputeNGrams(data, points, bigramDimensionMap); // Turn strings into numeric features.  
    Row<size_t> predictedClasses;  
    model.Classify(points, predictedClasses); // Stores output class IDs in 'predictedClasses'.  
  
    // Turn predictions back into string classes.  
    predictions.clear();  
    for (size_t i = 0; i < predictedClasses.n_elem; ++i)  
        predictions.push_back(classNames[predictedClasses[i]]);  
}
```

# serialize() via Cereal

```
// This function is used by Cereal to save/load the model.  
// We just serialize each of the members to 'ar'.  
template<typename Archive>  
void NGramLangDetectModel::serialize(Archive& ar, const unsigned int version /* unused */)  
{  
    ar(CEREAL_NVP(model));  
    ar(CEREAL_NVP(bigramDimensionMap));  
    ar(CEREAL_NVP(classNames));  
}
```



# Training program

```
#include <mlpack.hpp>
#include "ngram_langdetect_model.hpp"

using namespace std; using namespace arma; using namespace mlpack;

int main(int argc, char** argv)
{
    // blah blah blah, parse input options, load data, train/test split...

    NGramLangDetectModel m; // Create and train the model.
    m.Train(trainData, trainLabels, bigramsToKeep);

    // Now compute training and test accuracies.
    Row<size_t> trainPredictions, testPredictions;
    m.Classify(trainData, trainPredictions);
    m.Classify(testData, testPredictions);
    double trainAccuracy = 100.0 * ((double) accu(trainPredictions == trainLabels)) / trainLabels.n_elem;
    double testAccuracy = 100.0 * ((double) accu(testPredictions == testLabels)) / testLabels.n_elem;
    cout << "Training set accuracy: " << trainAccuracy << "%" << endl;
    cout << "Test set accuracy:      " << testAccuracy << "%" << endl;

    data::Save(modelFile, "model", m, true /* fatal on failure */);
}
```

# Training program

```
#include <mlpack.hpp>
#include "ngram_langdetect_model.hpp"

using namespace std; using namespace arma; using namespace mlpack;

int main(int argc, char** argv)
{
    // blah blah blah, parse input options, load data, train/test split...
```

```
NGram
m.Train
```

Compilation is trivial:

```
$ g++ -O3 -o ngram_langdetect_train ngram_langdetect_train.cpp -larmadillo
```

```
// No
```

```
Row<Size_t> trainPredictions, testPredictions,
```

```
m.Classify(trainData, trainPredictions);
```

```
m.Classify(testData, testPredictions);
```

```
double trainAccuracy = 100.0 * ((double) accu(trainPredictions == trainLabels)) / trainLabels.n_elem;
```

```
double testAccuracy = 100.0 * ((double) accu(testPredictions == testLabels)) / testLabels.n_elem;
```

```
cout << "Training set accuracy: " << trainAccuracy << "%" << endl;
```

```
cout << "Test set accuracy:      " << testAccuracy << "%" << endl;
```

```
data::Save(modelFile, "model", m, true /* fatal on failure */);
```

```
}
```

# Training program

```
#include <mlpack.hpp>
#include "ngram_langdetect_model.hpp"

using namespace std; using namespace arma; using namespace mlpack;

int main(int argc, char** argv)
{
    // blah blah blah. parse input options. load data. train/test split...

    Then run it:
NGramLangModel m.Train $ ./ngram_langdetect_train all_lang_data.csv all_lang_data.labels.csv
                           all_lang_data_model.bin
// Note Training set accuracy: 96.4473%.
Row<String> Test set accuracy:      96.3722%.
m.Classify();
m.Classify(resData, testPredictions),
double trainAccuracy = 100.0 * ((double) accu(trainPredictions == trainLabels)) / trainLabels.n_elem;
double testAccuracy = 100.0 * ((double) accu(testPredictions == testLabels)) / testLabels.n_elem;
cout << "Training set accuracy: " << trainAccuracy << "%." << endl;
cout << "Test set accuracy:      " << testAccuracy << "%." << endl;

data::Save(modelFile, "model", m, true /* fatal on failure */);
}
```

# Classification program

```
#include <mlpack.hpp>
#include "ngram_langdetect_model.hpp"

using namespace std; using namespace arma; using namespace mlpack;

int main(int argc, char** argv)
{
    // The first argument must be the model file. (Error checking removed for slides...)
    NGramLangDetectModel m;
    data::Load(string(argv[1]), "model", m, true /* fatal */);

    while(true)
    {
        string line;
        getline(cin, line);
        if (cin.eof())
            return 0;

        // Classify the point and print the predicted language.
        cout << m.Classify(line) << endl;
    }
}
```

# **Demo time**

This will never work!

# Container size initial results

```
$ docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
langdetect_pytorch	latest	5b20c970aac9	39 seconds ago	6.48GB
langdetect_py_full	latest	2591cd3cc86d	34 minutes ago	1.28GB
langdetect_py_slim	latest	308af891fc36	5 minutes ago	394MB
langdetect_py_alpine	latest	4534971adb18	About a minute ago	327MB

# Container size initial results

```
$ docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
langdetect_pytorch	latest	5b20c970aac9	39 seconds ago	6.48GB
langdetect_py_full	latest	2591cd3cc86d	34 minutes ago	1.28GB
langdetect_py_slim	latest	308af891fc36	5 minutes ago	394MB
langdetect_py_alpine	latest	4534971adb18	About a minute ago	327MB
langdetect_cpp_dynamic	latest	76c949b5ee22	3 hours ago	47.1MB

# Compilation reformation

- When we link dynamically against OpenBLAS, we have to keep every single OpenBLAS symbol... most of which we don't use.
- So we should link statically, and pass a number of options to gcc to strip unneeded code.

# Compilation reformation

- When we link dynamically against OpenBLAS, we have to keep every single OpenBLAS symbol... most of which we don't use.
- So we should link statically, and pass a number of options to gcc to strip unneeded code.

```
g++ -Os -DNDEBUG -DARMA_DONT_USE_WRAPPER -DARMA_NO_DEBUG -fopenmp \
    -fdata-sections -ffunction-sections -fwhole-program \
    -fvisibility=hidden -fvisibility-inlines-hidden \
    -static-libgcc \
    -static-libstdc++ \
    ngram_langdetect_run.cpp -o ngram_langdetect_run \
    -Wl,-Bstatic -lopenblas \
    -Wl,--as-needed -Wl,--gc-sections -fno-tie
```

# More container size results

```
$ docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
langdetect_pytorch	latest	5b20c970aac9	39 seconds ago	6.48GB
langdetect_py_full	latest	2591cd3cc86d	34 minutes ago	1.28GB
langdetect_py_slim	latest	308af891fc36	5 minutes ago	394MB
langdetect_py_alpine	latest	4534971adb18	About a minute ago	327MB
langdetect_cpp_dynamic	latest	76c949b5ee22	3 hours ago	47.1MB
langdetect_cpp_static	latest	73eb70105ffd	3 hours ago	25.5MB

## **Just a little bit more...**

- It turns out OpenBLAS is actually pretty large; linking against (singlethreaded) reference BLAS and LAPACK can save a lot of size.
- We can also use the distroless container to save even more space.

# More container size results

\$ docker images	REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
	langdetect_pytorch	latest	5b20c970aac9	39 seconds ago	6.48GB
	langdetect_py_full	latest	2591cd3cc86d	34 minutes ago	1.28GB
	langdetect_py_slim	latest	308af891fc36	5 minutes ago	394MB
	langdetect_py_alpine	latest	4534971adb18	About a minute ago	327MB
	langdetect_cpp_dynamic	latest	76c949b5ee22	3 hours ago	47.1MB
	langdetect_cpp_static	latest	73eb70105ffd	3 hours ago	25.5MB
	langdetect_cpp_distroless	latest	d69c6693cae8	34 seconds ago	3.96MB

# More container size results

\$ docker images	REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
	langdetect_pytorch	latest	5b20c970aac9	39 seconds ago	6.48GB
	langdetect_py_full	latest	2591cd3cc86d	34 minutes ago	1.28GB
	langdetect_py_slim	latest	308af891fc36	5 minutes ago	394MB
	langdetect_py_alpine	latest	4534971adb18	About a minute ago	327MB
	langdetect_cpp_dynamic	latest	76c949b5ee22	3 hours ago	47.1MB
	langdetect_cpp_static	latest	73eb70105ffd	3 hours ago	25.5MB
	langdetect_cpp_distroless	latest	d69c6693cae8	34 seconds ago	3.96MB

Hello world to langdetect, Python: 51.8MB → 327MB

Hello world to langdetect, C++: 2.7MB → 3.96MB

# More container size results

\$ docker images	REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
	langdetect_pytorch	latest	5b20c970aac9	39 seconds ago	6.48GB
	langdetect_py_full	latest	2591cd3cc86d	34 minutes ago	1.28GB
	langdetect_py_slim	latest	308af891fc36	5 minutes ago	394MB
	langdetect_py_alpine	latest	4534971adb18	About a minute ago	327MB
	langdetect_cpp_dynamic	latest	76c949b5ee22	3 hours ago	47.1MB
	langdetect_cpp_static	latest	73eb70105ffd	3 hours ago	25.5MB
	langdetect_cpp_distroless	latest	d69c6693cae8	34 seconds ago	3.96MB

Hello world to langdetect, Python: 51.8MB → 327MB

Hello world to langdetect, C++: 2.7MB → 3.96MB

And the actual program and model:

```
$ ls -lh
-rw-r--r-- 1 ryan ryan 176K Oct 31 14:46 all_lang_data_model.bin
-rwxr-xr-x 1 ryan ryan 1.5M Oct 31 14:53 ngram_langdetect_run_static
```

There are a lot of additional size optimization possibilities...

# **Conclusion**

1. You can do data science in C++. There is a robust ecosystem.

# Conclusion

1. You can do data science in C++. There is a robust ecosystem.
2. When you use C++, things go fast and bloat is way reduced.

# Conclusion

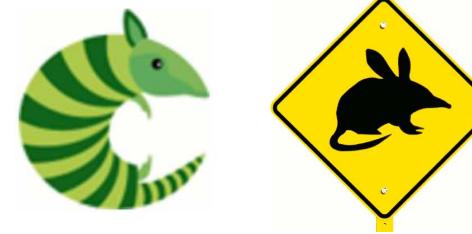
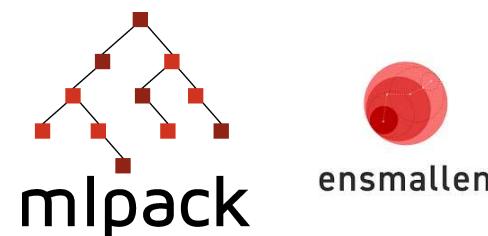
1. You can do data science in C++. There is a robust ecosystem.
2. When you use C++, things go fast and bloat is way reduced.
3. You should be doing data science deployments in C++. (*Especially: IoT, MPUs, low-resource, constrained environments*)

# Conclusion

1. You can do data science in C++. There is a robust ecosystem.
2. When you use C++, things go fast and bloat is way reduced.
3. You should be doing data science deployments in C++. (*Especially: IoT, MPUs, low-resource, constrained environments*)
4. The size footprints of C++ code are trivially 10x-100x smaller than equivalent Python workflows.

# Conclusion

1. You can do data science in C++. There is a robust ecosystem.
2. When you use C++, things go fast and bloat is way reduced.
3. You should be doing data science deployments in C++. (*Especially: IoT, MPUs, low-resource, constrained environments*)
4. The size footprints of C++ code are trivially 10x-100x smaller than equivalent Python workflows.
5. Check it out:
  - <https://www.mlpack.org>
  - <https://arma.sourceforge.net>
  - [https://github.com/rcurtin/lightweight\\_cpp\\_talk\\_code](https://github.com/rcurtin/lightweight_cpp_talk_code)
  - <https://www.ensmallen.org>
  - <https://coot.sourceforge.net>



# Questions and comments?



<http://www.ratml.org/>  
ryan@ratml.org