Hugot: huggingface 🤗 transformers in Golang

Riccardo Pinosio, Knights Analytics 

https://github.com/knights-analytics/hugot

Maintainers

HEAD R&D

Riccardo Pinosio

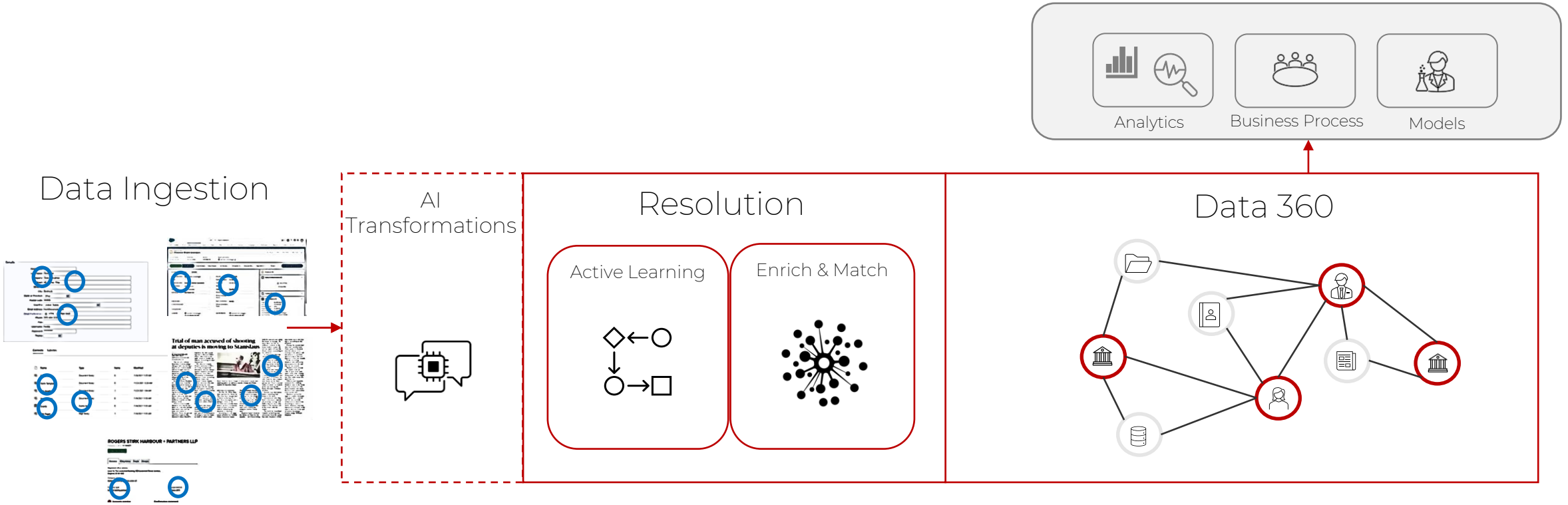
PhD & Lecturer in Machine Learning and 8 years delivering AI solutions

CTO

Rob Keevil

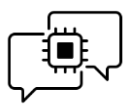
16 years architecting commercial software solutions in leadership and C level positions.

Alchemia Platform



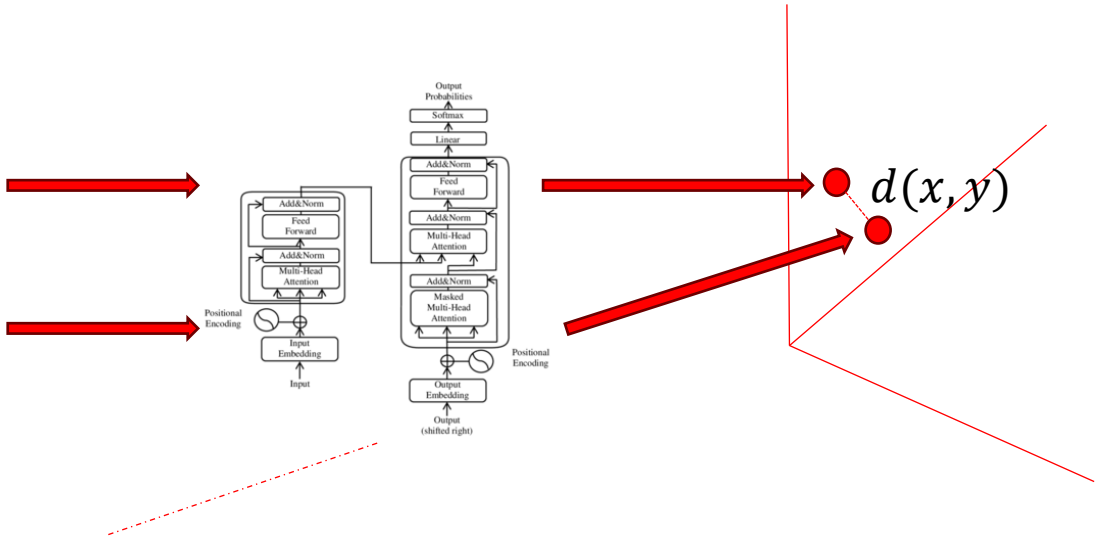
AI transformations

AI Transformations



"Sam Altman fired as CEO of ChatGPT maker OpenAI in shock move" (euronews)

"Sam Altman fired as CEO of OpenAI" (the Verge)



Tokenization

"Sam Altman fired as CEO of ChatGPT maker OpenAI in shock move"

encoding →

[101, 3520, 12456, 2386, 5045, 2004, 5766, 1997, 11834, 21600, 2102, 9338, 2330, 4886, 1999, 5213, 2693, 102]

← decoding

['[CLS]', 'sam', 'alt', '##man', 'fired', 'as', 'ceo', 'of', 'chat', '##gp', '##t', 'maker', 'open', '##ai', 'in', 'shock', 'move', '[SEP]']

Inference

[101, 3520, 12456, 2386, 5045, 2004, 5766, 1997, 11834, 21600, 2102, 9338, 2330, 4886, 1999, 5213, 2693, 102]

Transformer inference →

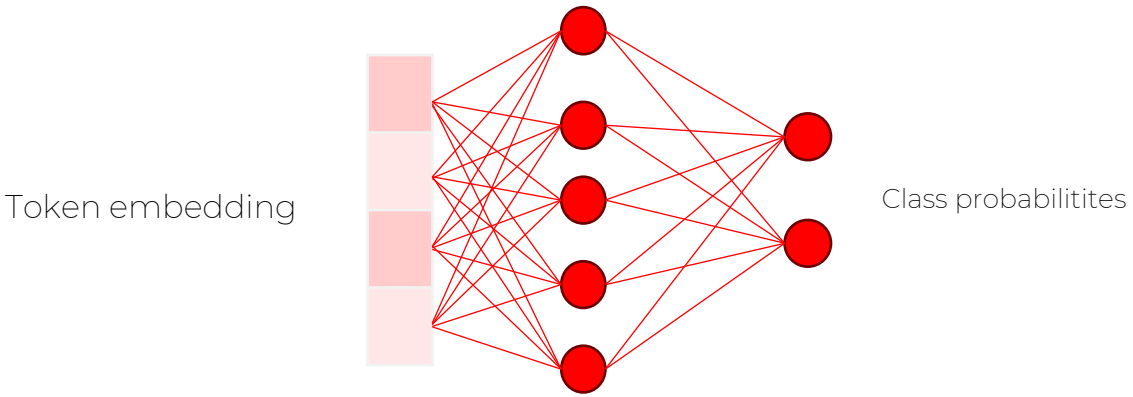
[-0.393, -0.149, -0.541, 0.245, -0.371, ...

How can we run these models in python?

- In Python, we can train transformers and run batch inference using the industry standard **huggingface** 🤗 and **optimum** libraries

- Easily load a transformer model as a **pipeline** and perform tokenization, inference and **postprocessing** with a trained model:

```
>>> from transformers import pipeline

>>> token_classifier = pipeline(model="Jean-Baptiste/camembert-ner", aggregation_strategy="simple")
>>> sentence = "Je m'appelle jean-baptiste et je vis à montréal"
>>> tokens = token_classifier(sentence)
>>> tokens
[{'entity_group': 'PER', 'score': 0.9931, 'word': 'jean-baptiste', 'start': 12, 'end': 26}, {'entity_g
```



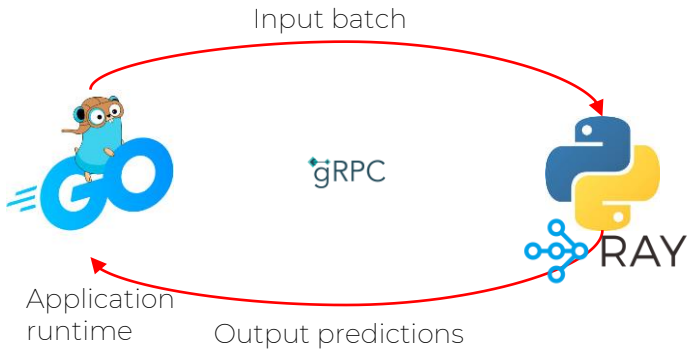
How can we run these models in go?

Native go

- Gorgonia

- Cybertron

Python via gRPC



Other non-standard libraries

go-huggingface
hfapigo

!

Call the huggingface API

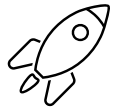
sugarme/transformers

Native go, “inspired by” huggingface transformers

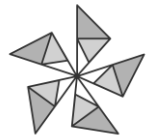
Enter hugot! (118★)

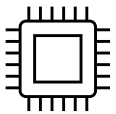
- Hugot (HUggingface GOlang Transformers)
- Library principles

Fidelity to the original Huggingface **pipelines**, so that models trained and tested in python can be seamlessly deployed in golang



Hassle-free and performant production use: we exclusively support **onnx** exports of huggingface models.





Run on your hardware: run transformer models tightly coupled with your go applications

Hugot pipelines

```go
import (
    "github.com/knights-analytics/hugot"
    "github.com/knights-analytics/hugot/pipelines"
)

// start a new session. This looks for the onnxruntime.so library in its default path,
// e.g. /usr/lib/onnxruntime.so
session, err := hugot.NewSession()

check(err)

defer func(session *hugot.Session) {
    err := session.Destroy()
    check(err)
}(session)

modelPath, err := session.DownloadModel("KnightsAnalytics/distilbert-base-uncased-finetuned-sst-2-english",
"./", hugot.NewDownloadOptions())
check(err)

// we now create the configuration for the text classification pipeline we want to create.
// Options to the pipeline can be set here using the Options field
config := TextClassificationConfig{
    ModelPath: modelPath,
    Name:      "testPipeline",
}
// then we create out pipeline.
// Note: the pipeline will also be added to the session object so all pipelines can be destroyed at once
sentimentPipeline, err := NewPipeline(session, config)
check(err)

// we can now use the pipeline for prediction on a batch of strings
batch := []string{"This movie is disgustingly good !", "The director tried too much"}
batchResult, err := sentimentPipeline.RunPipeline(batch)
check(err)
s, err := json.Marshal(batchResult)
check(err)
fmt.Println(string(s))
//
{"ClassificationOutputs":[[{"Label":"POSITIVE","Score":0.9998536}],[{"Label":"NEGATIVE","Score":0.99752176}]]}
```

Hugot cli: transformers **without** python **and** go!

```
$ hugot run \
--model=KnightsAnalytics/distilbert-base-uncased-finetuned-
sst-2-english \
--input=/path/to/input.jsonl \
--output=/path/to/folder/output \
--type=textClassification
```

Hugot pipelines

Currently implemented: ✓

- CPU inference for
  - FeatureExtractionPipeline
  - TokenClassificationPipeline
  - TextClassificationPipeline

- Pipelines and session object are thread-safe for inference (can be called through goroutines/channels)

In the works! 

- Object detection
- Feature extraction from images
- GPU inference (CUDA) through onnxruntime

The pipeline abstraction

```go
type TokenClassificationPipeline struct {
    BasePipeline
    IdLabelMap          map[int]string
    AggregationStrategy string
    IgnoreLabels        []string
}
```

embeds

```go
type BasePipeline struct {
    ModelPath        string
    OnnxFilename     string
    PipelineName     string
    OrtSession       *ort.DynamicAdvancedSession
    OrtOptions       *ort.SessionOptions
    Tokenizer        *tokenizers.Tokenizer
            …
}
```

implements

```go
type Pipeline interface {
    Destroy() error
    GetStats() []string
    GetOutputDim() int
    Validate() error
    Run([]string) (PipelineBatchOutput, error)
}
```

Implements: PipelineBatchOutput interface

```go
func (p *TokenClassificationPipeline) Run(inputs []string) (*TokenClassificationOutput, error) {
    batch := p.Preprocess(inputs)
    batch, errForward := p.Forward(batch)
    if errForward != nil {
        return nil, errForward
    }
    return p.Postprocess(batch)
}
```

Performs tokenization, binding to **Rust** tokenizers via CGO

Performs forward inference with **ONNXruntime** model
[go_onnxruntime]

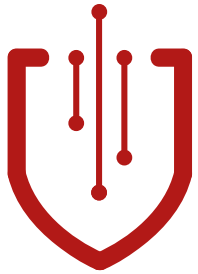
Postprocesses the logits to turn them into the structured output

The session object

```go
type Session struct {
    featureExtractionPipelines   pipelineMap[*pipelines.FeatureExtractionPipeline]
    tokenClassificationPipelines pipelineMap[*pipelines.TokenClassificationPipeline]
    textClassificationPipelines  pipelineMap[*pipelines.TextClassificationPipeline]
    ortOptions                   *ort.SessionOptions
}

func NewPipeline[T pipelines.Pipeline](s *Session, pipelineConfig pipelines.PipelineConfig[T]) (T, error) {
    var pipeline T
    var err error

[…]

    switch any(pipeline).(type) {
    case *pipelines.TokenClassificationPipeline:
        config := any(pipelineConfig).(pipelines.PipelineConfig[*pipelines.TokenClassificationPipeline])
        pipelineInitialised, err := pipelines.NewTokenClassificationPipeline(config, s.ortOptions)
        if err != nil {
            return pipeline, err
        }
        s.tokenClassificationPipelines[config.Name] = pipelineInitialised
        pipeline = any(pipelineInitialised).(T)
    case *pipelines.TextClassificationPipeline:
            …
    return pipeline, err
}
```

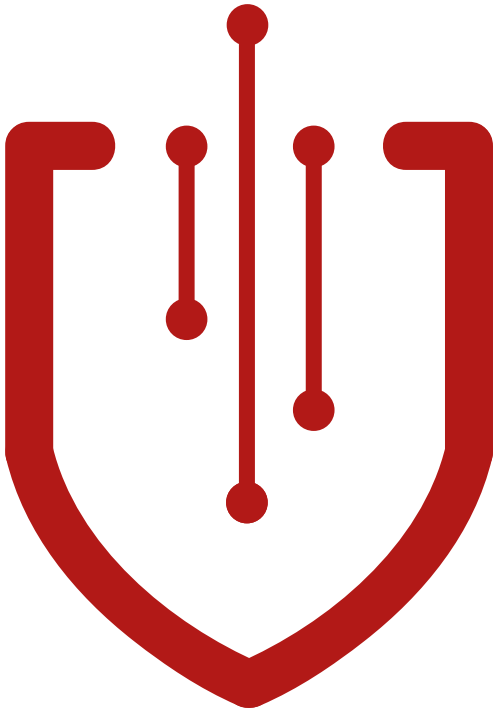
No support for generics in struct methods: https://go.googlesource.com/proposal/+/refs/heads/master/design/43651-type-parameters.md#No-parameterized-methods

Conclusions



- Hugot ★ is a go library to run onnxruntime transformer pipelines tightly coupled with golang applications
- It is thread-safe and leverages Rust tokenizers and onnxruntime for stability and performance
- It implements 3 fundamental model pipelines, with more to come
- Inference is on CPU, but GPU will be supported via onnxruntime
- Used in production, extensive benchmarks in the works

COMPLEXITY. SIMPLIFIED.



hello@knightsanalytics.com

www.knightsanalytics.com

COMPLEXITY. SIMPLIFIED.