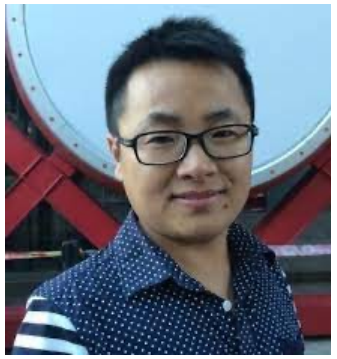
Ken

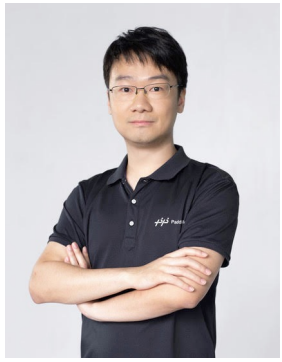
Valia

Gary

Ernie



Yanjun



Zeyu

ACL-2022 Tutorial:
Part A: Simple Uses of Deep Nets

Ken Church, Baidu, USA

Valia Kordoni, Humboldt-Universitaet zu Berlin, Germany

Gary Marcus, Robust.AI

Ernest Davis, NYU

Yanjun MA, Baidu, China

Zeyu Chen, Baidu, China

Outline

- Part A: Glass is half-full
  - Deep nets can do much
- Part B: Glass is half-empty
  - There is always more work to do

Make deep nets accessible to masses (including non-programmers)

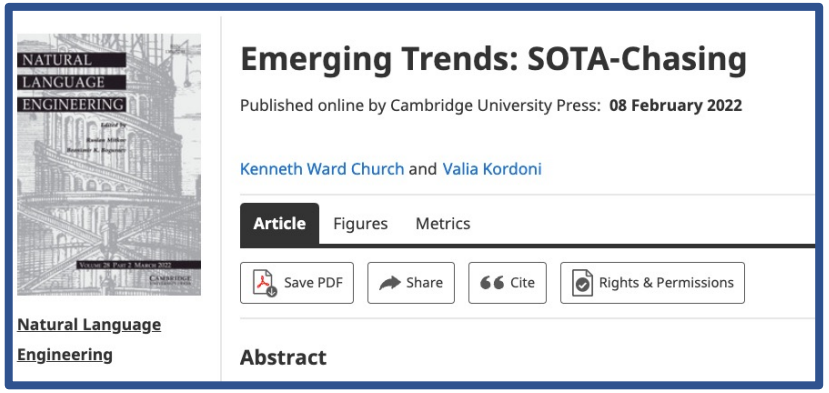
Advocate combo of Part A with
1. AI Knowledge Representation
2. Linguistics and Philosophy

- GFT Code,
- 100s of examples,
- slides, videos, papers

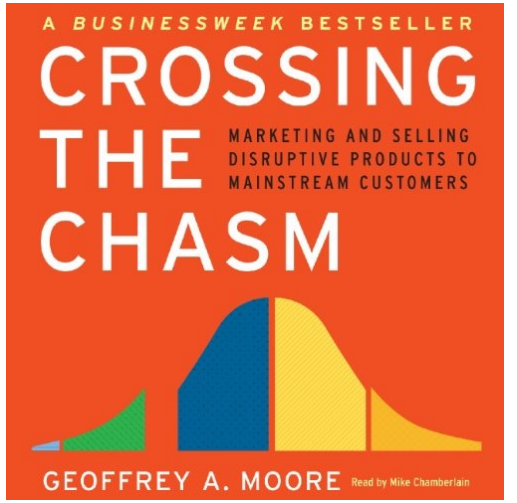
Agenda: Part A

- **Strategy**
- *gft* Cheatsheet
  - 4-5 Functions + 4 Arguments
  - Amazing how much can be done with so little
- Seven Simple Examples
- More Tasks: MT, ASR, Vision
- Two Questions
  - How do we *find* the good stuff?
  - How do we *use* the good stuff?
- Why de-emphasize pre-training?
- Conclusions

Strategy: Inclusiveness

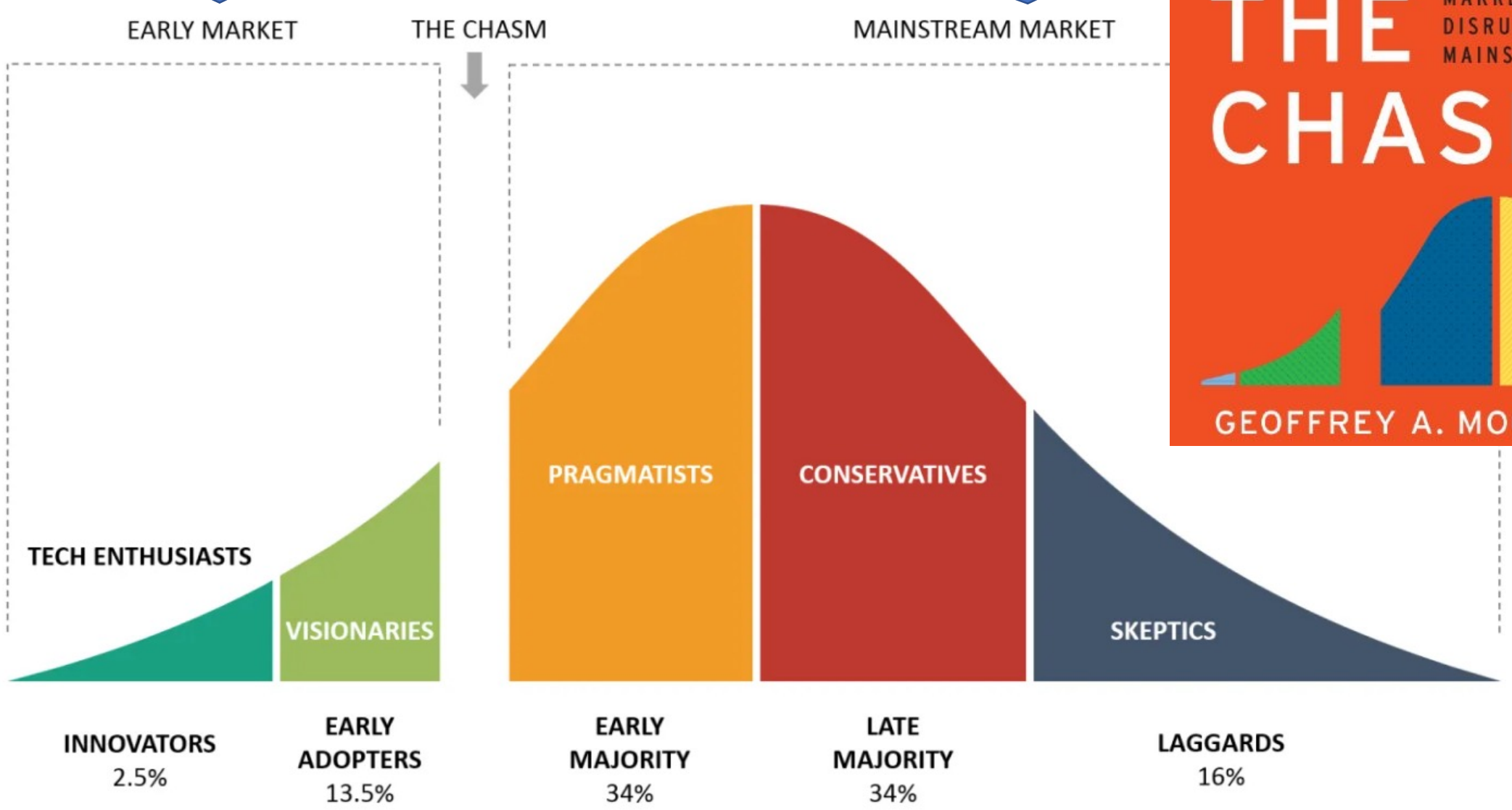


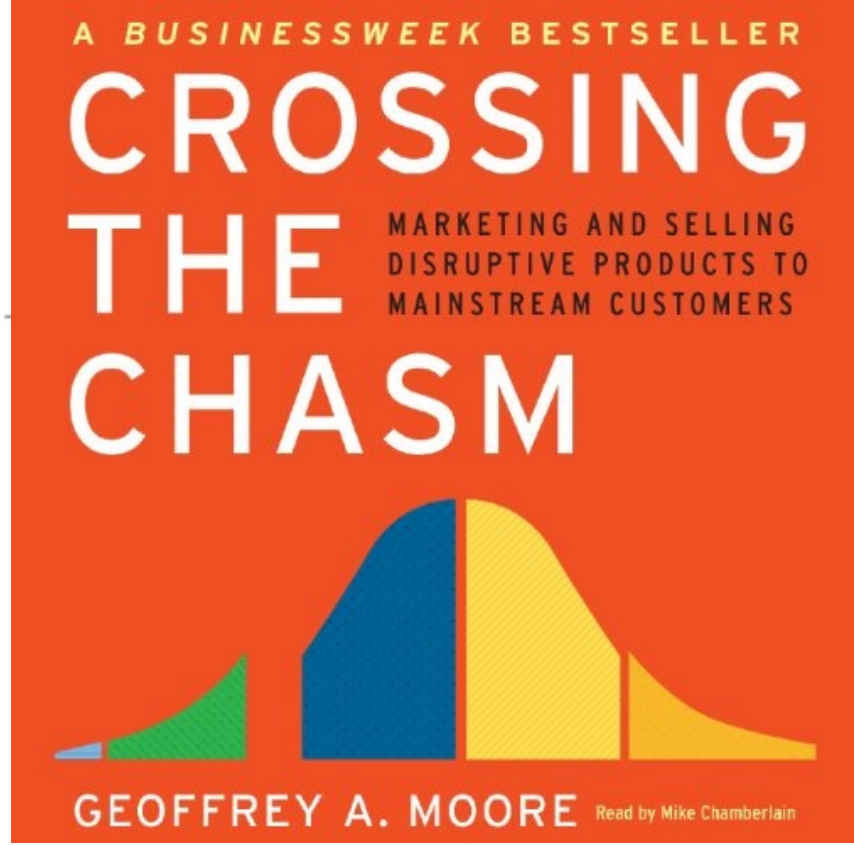
- Get your priorities right: Inclusiveness >> Exclusiveness
  - SOTA-Chasing 👎 (Exclusive: mine is bigger than yours)
  - Crossing the Chasm 👍 (Inclusive: Appeal to mainstream users)
  - Hubs: Get big quick 👍 (Inclusive: Usage >> SOTA)



Engineers

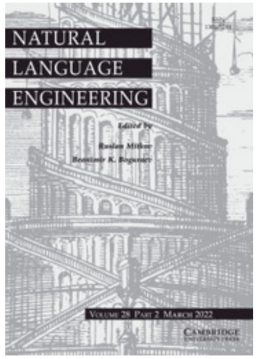
Bigger Opportunities:
(I)users (aka poets)





Poets

- Poets is an imperfect metaphor,
  - intended as a gesture toward inclusion.
- The future for deep nets
  - will benefit by reaching out to
    - a broad audience of potential users,
  - including people with
    - little or no programming skills,
    - and little interest in training models.
- Lessons from success of Unix
  - Less is more
  - Inclusion: Appeal to power users as well as masses
  - Portability: Inclusiveness++



Natural Language Engineering

Emerging trends: Deep nets for poets

Published online by Cambridge University Press: **01 September 2021**

Kenneth Ward Church, Xiaopeng Yuan, Sheng Guo, Zewu Wu, Yehua Yang and Zeyu Chen

Article | Figures | Metrics

Save PDF | Share | Cite | Rights & Permissions

h/ACL2

Unix™ for Poets

Kenneth Ward Church
AT&T Bell Laboratories
kwc@research.att.com

- Text is available like never before
  - Dictionaries, corpora, etc.
  - Data Collection Efforts:
    ACL/DCI, BNC, CLR, ECI, EDR, ICAME, LDC
  - Information Super Highway Roadkill:
    email, bboards, faxes
  - Billions and billions of words
- What can we do with it all?
- It is better to do something simple,
  than nothing at all.
- You can do the simple things yourself
  (DIY is more satisfying than begging for ‘‘help’’ from a computer officer.)

(Sensible) Priority for Startups

Hub Strategy: Usage >> SOTA

(Mistaken) Priority Research

- Startup Strategy:
  - Get big quick
- SOTA is like sprinting (hard work)
  - Avoid temptation to sprint to front
    - (except when it is worth the effort)
  - Hubs spend most of the race
    - in the peloton drafting researchers
  - Make it easy for 3rd parties
    - to post their best models on your hub
- Use of 3rd parties → Wide range in quality
  - Best are amazing: best in class, industrial strength…
  - But some don't work well, and may not even load…
  - Help users find good stuff

erminology

0-shot

Unsupervised

Supervised

Few-shot

Standard Terminology	Proposed Alternatives
~~Base/Foundation/Pre-Trained~~	f_{pre}
~~Fine-Tuning~~	*fit*
~~Inference~~	*predict*

Emphasize this

De-emphasize this

Teaser: Stay-tuned

gft (general fine-tuning):
A Little Language for Deep Nets
(Unix Philosophy: *Less is More*)

Standard 3-Step Recipe

Step	gft Support	Description	Time	Hardware
1		Pre-Training	Days/Weeks	Large GPU Cluster
2	*gft_fit*	Fine-Tuning	Hours/Days	1+ GPUs
3	*gft_predict*	Inference	Seconds/Minutes	0+ GPUs

- Terminology borrowed from *sklearn*:
  - *fit*: $f_{pre} + data \rightarrow f_{post}$
  - *predict*: $f(x) \rightarrow \hat{y}$
- *fit* and *predict* are (almost) all you need
  - *gft* programs are short (1-line)
  - No (not much) programming required
    - No python in this tutorial
    - Examples on hubs are (unnecessarily) long/complicated

Examples of 1-line GFT Programs

Step 2: *gft_fit*

```
gft_fit --eqn 'classify: label ~ text' \
    --model H:bert-base-cased \
    --data H:emotion \
    --output_dir $outdir
```

Data

f_{pre}: Pre-trained Model

f_{post}: Post-trained Model

Step 3: *gft_predict*

```
# text-classification: sentiment analysis
echo 'I love you.' | gft_predict --task text-classification
# I love you.   POSITIVE    0.9998705387115479
```

x $\hat{y}$ Score

Little Languages (from Unix)

https://comp590-19s.github.io/

Kris Jordan (UNC Chapel Hill, 2019)

Bentley (CACM-1986)

programming pearls

by Jon Bentley

LITTLE LANGUAGES

When you say "language," most programmers think of the big ones, like FORTRAN or COBOL or Pascal. In fact, a language is any mechanism to express intent, and the input to many programs can be viewed profitably as statements in a language. This column is about those "little languages."

Programmers deal with microscopic languages every day. Consider printing a floating-point number in six characters, including a decimal point and two subsequent digits. Rather than writing a subroutine for the task, a FORTRAN programmer specifies the format `F6.2`, and a COBOL programmer defines the picture `999.99`. Each of these descriptions is a statement in a well-defined little language. While the languages are quite different, each is appropriate for its problem domain: although a FORTRAN programmer might complain that `999999.99999` is too long when `F12.5` could do the job, the COBOLer can't even express in FORTRAN such common financial patterns as `$,$$$,$$9.99`. FORTRAN is aimed at scientific computing, COBOL is designed for business.

In the good old days, real programmers would swagger to a key punch and, standing the whole time, crank out nine cards like:

```
//SUMMARY  JOB  REGION=(100K,50K)
//         EXEC PGM=SUMMAR
//SYSIN    DD   DSNAME=REP.8601,DISP=OLD,
//              UNIT=2314,SPACE=(TRK,(1,1,1)),
//              VOLUME=SER=577632
//SYSOUT   DD   DSNAME=SUM.8601,DISP=(,KEEP),
//              UNIT=2314,SPACE=(TRK,(1,1,1)),
//              VOLUME=SER=577632
//SYSABEND DD   SYSOUT=A
```

Today's young whippersnappers do this simple job by typing

```
summarize <jan.report >jan.summary
```

Modern successors to the old "job control" languages are not only more convenient to use, they are fundamentally more powerful than their predecessors. In the June column, for instance, Doug McIlroy implemented a program to find the *K* most common words in a document in six lines of the UNIX® SHELL language.

Languages surround programmers, yet many programmers don't exploit linguistic insights. Examining programs under a linguistic light can give you a better understanding of the tools you now use, and can teach you design principles for building elegant interfaces to your future programs. This column will show how the user interfaces to half a dozen interesting programs can be viewed as little languages.

This column is built around Brian Kernighan's PIC language for making line drawings. Its compiler is implemented on the UNIX system, which is particularly supportive (and exploitative) of language processing; the sidebar on pages 714–715 shows how little languages can be implemented in a more primitive computing environment (BASIC on a personal computer).

The next section introduces PIC and the following section compares it to alternative systems. Subsequent sections discuss little languages that compile into PIC and little languages used to build PIC.

The PIC Language

If you're talking about compilers, you might want to depict their behavior with a picture:

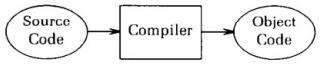
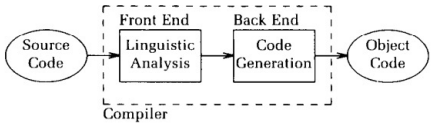
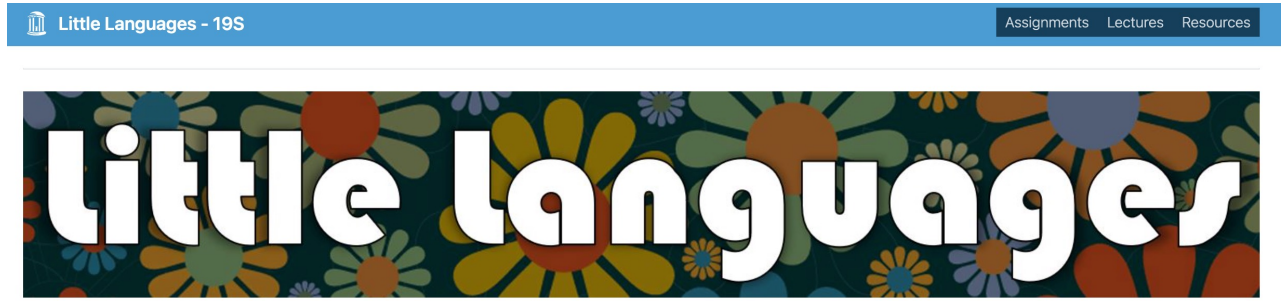


FIGURE 1. A Simple View of a Compiler

(This diagram is genuine PIC output; we'll see its input description shortly.) Or you may desire a little more detail about the internal structure:



UNIX is a trademark of AT&T Bell Laboratories.



Computer scientists and software engineers depend on tools that interpret Little Languages (Bentley, UNC '76) frequently in their day-to-day work: command-line shells, project build system configurations, regular expressions, utility programs, documentation markup languages, modal text editors, and so on. As these languages are numerous and domain-specific, our focus will be on their first principles, common challenges, implementations, and limitations.

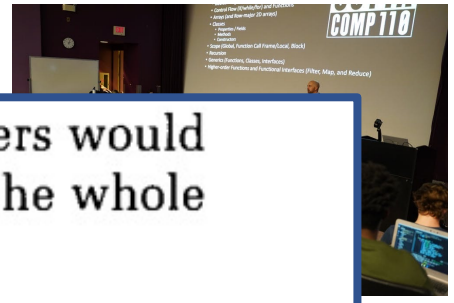
This course will explore the interpretation and applications of little languages through case studies on standard programming environment tools with historical significance, such as `grep`, `make`, `bash`, `vim`, and more. This course highlights important ideas in computational theory through direct, pragmatic applications in text pattern matching and structured language processing.

You will gain experience working in a systems language appropriate for implementing command-line tools with their own little languages. You will gain comfort working in a Unix-like programming environment and knowledge of its standard tooling and utilities. You will learn to make use of a Unix-like operating system's facilities and APIs for file system input/output, process management, and memory management.

In this course, you will:

1. Implement the essences of practical, historically significant Unix utilities, each with its own little language.
   - `dc` (1969) a calculator whose simple, reverse-polish notation language commands a virtual stack machine. It is the oldest little language in Unix.
   - `bc` (1975), the successor to `dc`, is an infix-based context free language needing a pushdown automata equivalent to parse. It will be implemented, as it was historically, as a frontend translator for `dc`.
   - `grep` (1974) is a regular expression pattern matching tool still widely used today. Its implementation will construct and simulate a non-deterministic finite state machine.
   - `sh` (1977) the interpretation of Bourne's shell, or command-line interpreter, introduces fundamental operating systems concepts.

Little Languages (from Unix)



In the good old days, real programmers would swagger to a key punch and, standing the whole time, crank out nine cards like:

```
//SUMMARY   JOB   REGION=(100K,50K)
//           EXEC  PGM=SUMMAR
//SYSIN     DD    DSNAME=REP.8601,DISP=OLD,
//                UNIT=2314,SPACE=(TRK,(1,1,1)),
//                VOLUME=SER=577632
//SYSOUT    DD    DSNAME=SUM.8601,DISP=(,KEEP),
//                UNIT=2314,SPACE=(TRK,(1,1,1)),
//                VOLUME=SER=577632
//SYSABEND  DD    SYSOUT=A
```

Today's young whippersnappers do this simple job by typing

```
summarize <jan.report >jan.summary
```

Bentley (CACM-1986)

programming pearls

by Jon Bentley

LITTLE LANGUAGES

When you say "language," most programmers think of the big ones, like FORTRAN or COBOL or Pascal. In fact, a language is any mechanism to express intent, and the input to many programs can be viewed profitably as statements in a language. This column is about those "little languages."

Programmers deal with microscopic languages every day. Consider printing a floating-point number in six characters, including a decimal point and two subsequent digits. Rather than writing a subroutine for the task, a FORTRAN programmer specifies the format `F6.2`, and a COBOL programmer defines the picture `999.99`. Each of these descriptions is a statement in a well-defined little language. While the languages are quite different, each is appropriate for its problem domain: although a FORTRAN programmer might complain that `999999.99999` is too long when `F12.5` could do the job, the COBOLer can't even express in FORTRAN such common financial patterns as `$,$$$,$$9.99`. FORTRAN is aimed at scientific computing, COBOL is designed for business.

In the good old days, real programmers would swagger to a key punch and, standing the whole time, crank out nine cards like:

```
//SUMMARY   JOB   REGION=(100K,50K)
//           EXEC  PGM=SUMMAR
//SYSIN     DD    DSNAME=REP.8601,DISP=OLD,
//                UNIT=2314,SPACE=(TRK,(1,1,1)),
//                VOLUME=SER=577632
//SYSOUT    DD    DSNAME=SUM.8601,DISP=(,KEEP),
//                UNIT=2314,SPACE=(TRK,(1,1,1)),
//                VOLUME=SER=577632
//SYSABEND  DD    SYSOUT=A
```

Today's young whippersnappers do this simple job by typing

```
summarize <jan.report >jan.summary
```

Modern successors to the old "job control" languages are not only more convenient to use, they are fundamentally more powerful than their predecessors. In the June column, for instance, Doug McIlroy implemented a program to find the *K* most common words in a document in six lines of the UNIX® SHELL language.

Languages surround programmers, yet many programmers don't exploit linguistic insights. Examining programs under a linguistic light can give you a better understanding of the tools you now use, and can teach you design principles for building elegant interfaces to your future programs. This column will show how the user interfaces to half a dozen interesting programs can be viewed as little languages.

This column is built around Brian Kernighan's PIC language for making line drawings. Its compiler is implemented on the UNIX system, which is particularly supportive (and exploitative) of language processing; the sidebar on pages 714–715 shows how little languages can be implemented in a more primitive computing environment (BASIC on a personal computer).

The next section introduces PIC and the following section compares it to alternative systems. Subsequent sections discuss little languages that compile into PIC and little languages used to build PIC.

The PIC Language

If you're talking about compilers, you might want to depict their behavior with a picture:

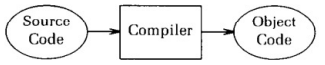
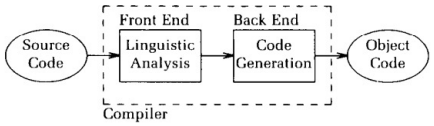


FIGURE 1. A Simple View of a Compiler

(This diagram is genuine PIC output; we'll see its input description shortly.) Or you may desire a little more detail about the internal structure:



UNIX is a trademark of AT&T Bell Laboratories.

We propose *gft*, a Little Language

In the good old days, real programmers would swagger to a key punch and, standing the whole time, crank out nine cards like:

```
//SUMMARY   JOB   REGION=(100K,50K)
//           EXEC  PGM=SUMMAR
//SYSIN     DD    DSNAME=REP.8601,DISP=OLD,
//                UNIT=2314,SPACE=(TRK,(1,1,1)),
//                VOLUME=SER=577632
//SYSOUT    DD    DSNAME=SUM.8601,DISP=(,KEEP),
//                UNIT=2314,SPACE=(TRK,(1,1,1)),
//                VOLUME=SER=577632
//SYSABEND  DD    SYSOUT=A
```

Today's young whippersnappers do this simple job by typing

```
summarize <jan.report >jan.summary
```

- In the good old days,
  - real programmers would copy a few hundred lines of PyTorch from HuggingFace
  - and modify as necessary to change
    - models,
    - datasets,
    - task,
    - etc.
- With *gft*, 100s of lines → 1-line
  - More accessible to masses
  - (including non-programmers)

gft Cheat Sheet
(General Fine-Tuning)

Amazing how much can be done with so little

4+1 Functions

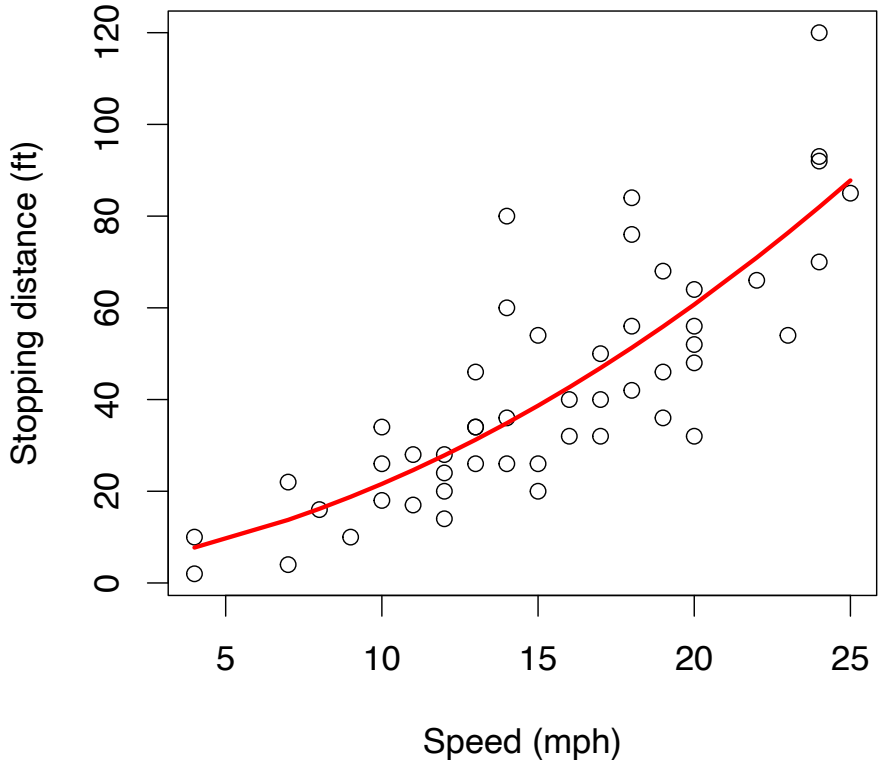
1. gft_fit: $f_{pre} \rightarrow f_{post}$ (fine-tuning)
   - 4 Arguments, --output_dir, --metric, --splits
   - (plus most args in most hubs)
2. gft_predict: $f(x) \rightarrow \hat{y}$ (inference)
   - Input: 4 Arguments (x from data or stdin)
   - Output: $\hat{y}$ for each x
3. gft_eval: Score model on dataset
   - Input: 4 Arguments, --split, --metric, …
   - Output: Score
4. gft_summary: Find good stuff
   - Input: 4 Arguments
   - (may include: __*contains*__, __*infer*__)
5. gft_cat_dataset: Output data to *stdout*
   - Input: 4 Arguments (--data, --eqn)

4 Arguments

- --data
- --model
- --eqn $task: y \sim x_1 + x_2$
- --task
   1. classify (text-classification)
   2. classify_tokens (token-classification)
   3. classify_spans (QA, question-answering)
   4. classify_audio (audio-classification)
   5. classify_images (image-classification)
   6. regress
   7. text-generation
   8. MT (translation)
   9. ASR (ctc, automatic-speech-recognition)
   10. fill-mask

gft (General Fine-Tuning) is based on *glm* (General Linear Models) in R

Make deep nets look like regression
(Not much programming)



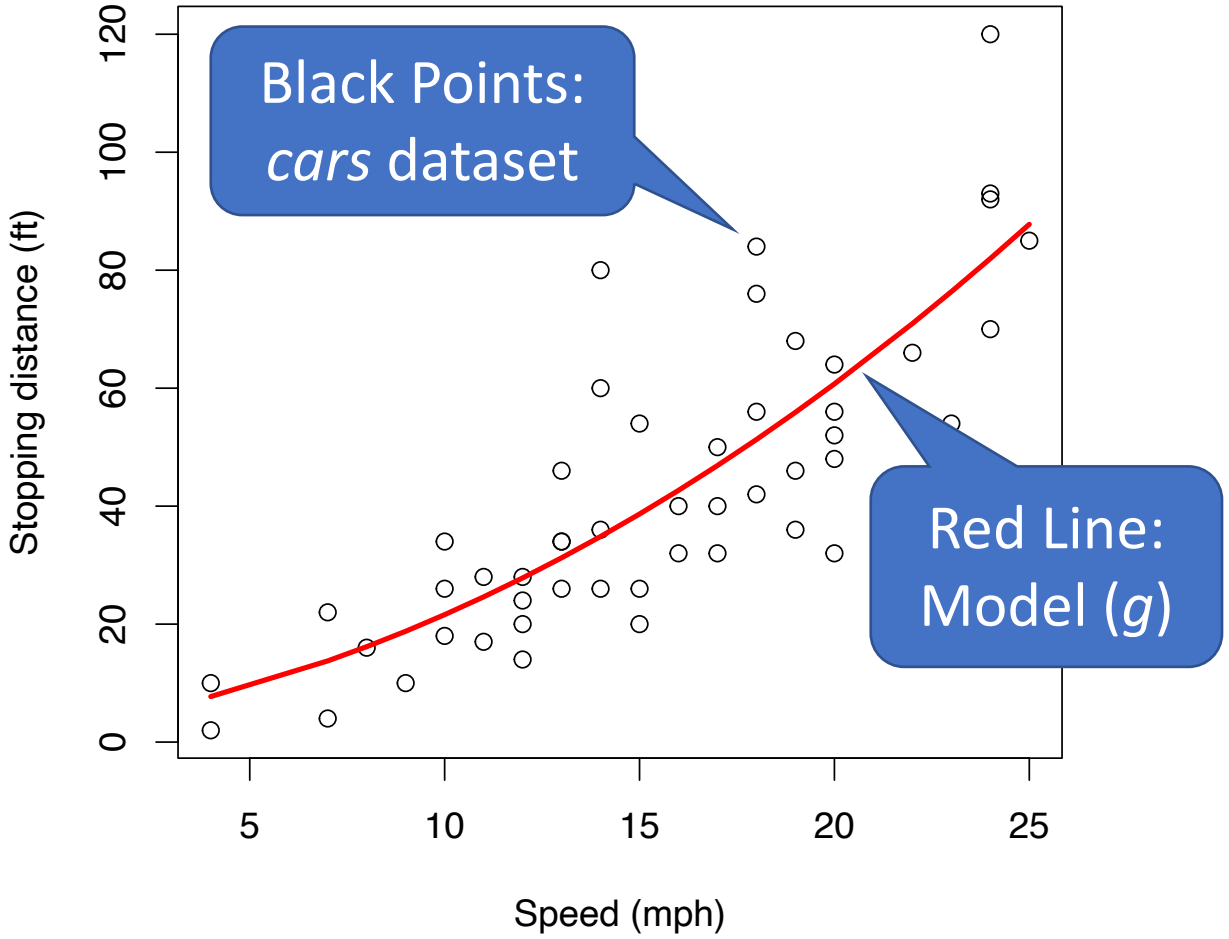
3.1 An Example of Fit and Predict in R

```r
# Summarize the cars dataset
summary(cars)
# Create the black points
plot(cars, xlab="Speed (mph)", ylab="Stopping distance (ft)")
# Fit a model g to cars dataset,
# assuming dist is a quadratic function of speed
g = glm(dist ~ poly(speed, 2), data=cars)
# Summarize the model g
summary(g)
o = order(cars$speed)
# Show predictions as a red line
lines(cars$speed[o], predict(g,cars)[o], col="red", lwd=3)
```

Listing 1. Example of fit and predict in R

Regression in *R*:

Summarize (almost) anything




```
> head(cars)
  speed dist
1     4    2
2     4   10
3     7    4
4     7   22
5     8   16
6     9   10
> summary(cars)
     speed           dist       
 Min.   : 4.0   Min.   :  2.00  
 1st Qu.:12.0   1st Qu.: 26.00  
 Median :15.0   Median : 36.00  
 Mean   :15.4   Mean   : 42.98  
 3rd Qu.:19.0   3rd Qu.: 56.00  
 Max.   :25.0   Max.   :120.00  
>
```

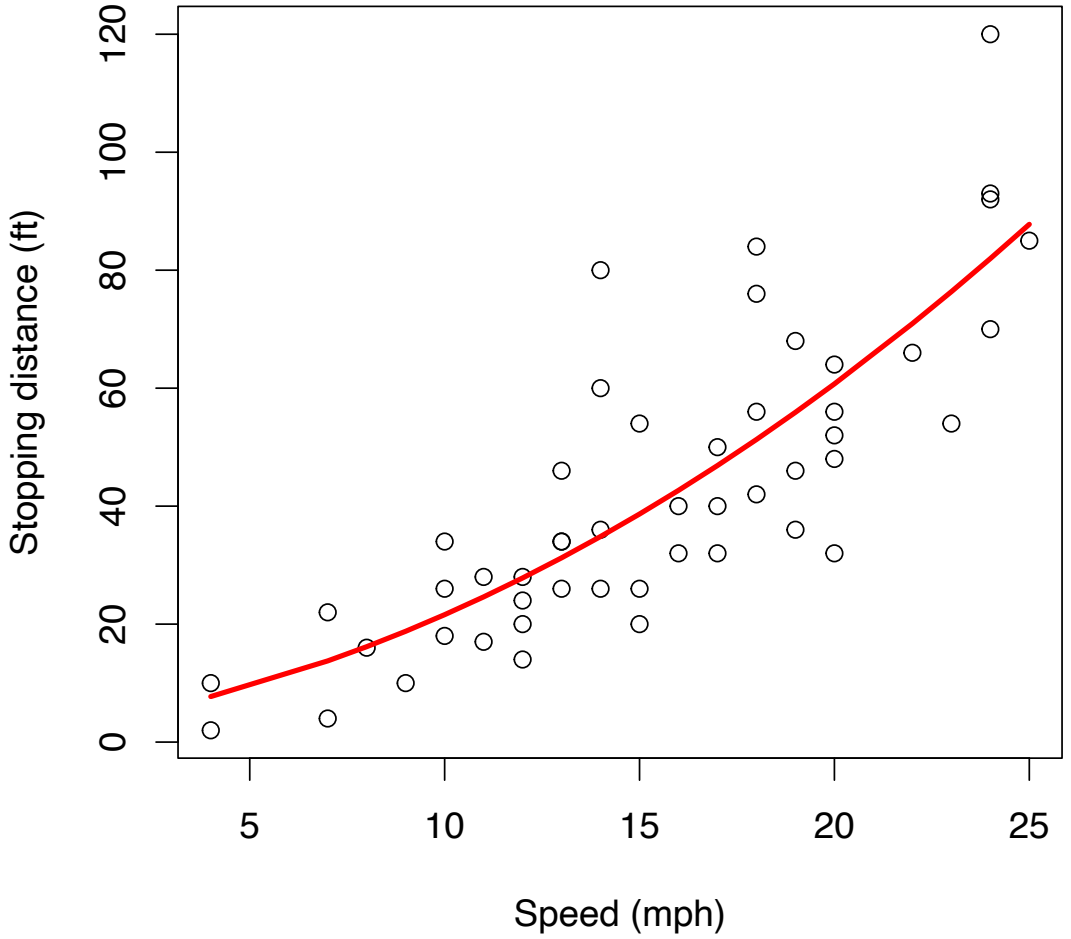
Summarize a dataset (*cars*)

Fit a model (*g*) to data (*cars*)

Regression in *R*:

Summarize (almost) anything

```
> plot(cars, xlab="Speed (mph)", ylab="Stopping distance (ft)")
> g = glm(dist ~ poly(speed,2), data=cars)
> o = order(cars$speed)
> lines(cars$speed[o], predict(g,cars)[o], col="red", lwd=3)
```



Regression in *R*:

Summarize (almost) anything

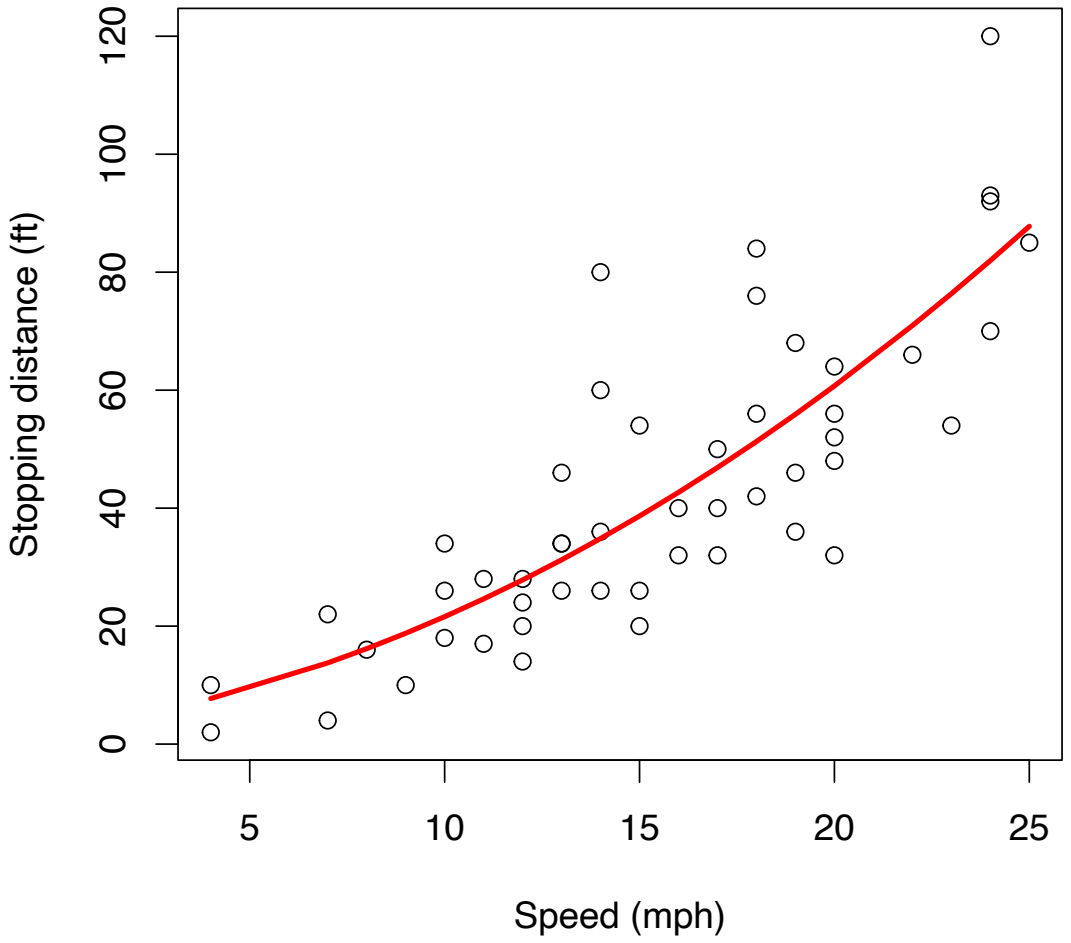
Fit a model (*g*) to data (*cars*)

Predict

Summarize a model (*g*)

Opportunity to Improve *g*

```
> plot(cars, xlab="Speed (mph)", ylab="Stopping distance (ft)")
> g = glm(dist ~ poly(speed,2), data=cars)
> o = order(cars$speed)
> lines(cars$speed[o], predict(g,cars)[o], col="red", lwd=3)
> summary(g)

Call:
glm(formula = dist ~ poly(speed, 2), data = cars)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-28.720   -9.184   -3.188    4.628   45.152

Coefficients:
                Estimate Std. Error t value Pr(>|t|)
(Intercept)       42.980      2.146  20.026  < 2e-16 ***
poly(speed, 2)1  145.552     15.176   9.591 1.21e-12 ***
poly(speed, 2)2   22.996     15.176   1.515    0.136
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for gaussian family taken to be 230.3131)

    Null deviance: 32539  on 49  degrees of freedom
Residual deviance: 10825  on 47  degrees of freedom
AIC: 418.77

Number of Fisher Scoring iterations: 2
```



What is ~~fine-tuning~~ *fit*?

- *fit*: $f_{pre} + data \rightarrow f_{post}$
- f_{pre}:
  - bert-base-cased model
  - from HuggingFace
- $data$:
  - Emotion dataset
  - from HuggingFace
- f_{post}:
  - Maps text to emotion labels

R Program

```
> plot(cars, xlab="Speed (mph)", ylab="Stopping distance (ft)")
> g = glm(dist ~ poly(speed,2), data=cars)
```

gft Program

```
gft_fit --eqn 'classify: label ~ text' \
    --model H:bert-base-cased \
    --data H:emotion \
    --output_dir $outdir
```

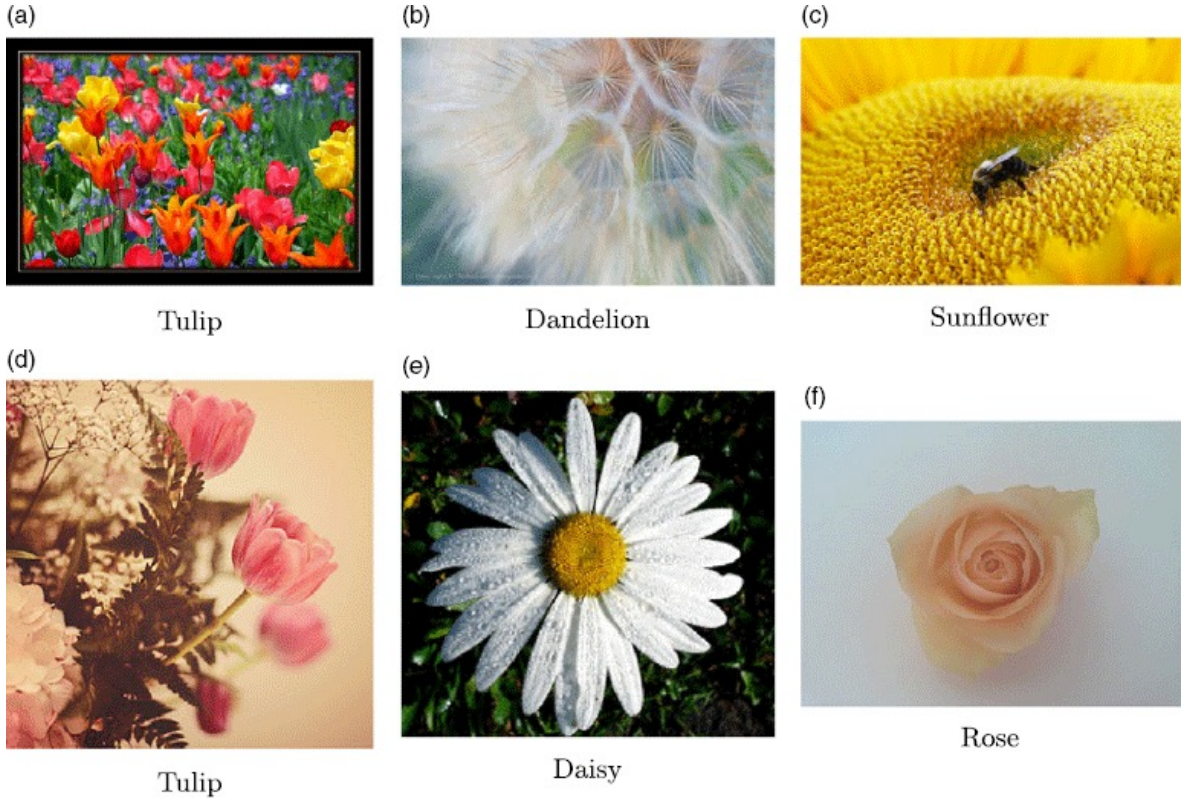
Pre-trained (foundation) model	Parameters	Training data
ResNet-50 (He *et al.* 2016)	23M	14M images from ImageNet
VT (Wu *et al.* 2020)	11.7–21.9M	14M images from ImageNet
Wav2vec (Baevski *et al.* 2020)	95–317M	960 hours from LibriSpeech
BERT (Devlin *et al.* 2019)	110–340M	3.3B words from Books/Wiki
ERNIE 2.0 (Sun *et al.* 2020)	110–340M	7.9B en + 15B zh tokens
ERNIE 3.0 (Sun *et al.* 2021)	10B	375B tokens of text, as well as knowledge graph

Dataset	Description
Billion Word Benchmark (Chelba *et al.* 2013)	A billion words of English
Common Crawl (Buck, Heafield, and van Ooyen 2014)	https://github.com/commoncrawl
Book Corpus (Zhu *et al.* 2015)	Speech with text
ImageNet (Deng *et al.* 2009)	14M images, annotated with 21k classes
LibriSpeech (Panayotov *et al.* 2015)	960 hours of speech with text
LJ Speech	https://keithito.com/LJ-Speech-Dataset/

What is ~~fine-tuning~~ *fit*?

- *fit*: $f_{pre} + data \rightarrow f_{post}$
- f_{pre}: ResNet-50
  - Trained on ImageNet (1k class labels)
  - Maps images to ImageNet Labels
- $data$: Flowers
  - Maps images to Flower Labels (5 class labels)
  - Flower Labels ≠ ImageNet Labels
- f_{post}:
  - maps images to Flower Labels

$data$ → Flowers



f_{pre}: Pre-trained Model

Pre-trained (foundation) model	Parameters	Training data
ResNet-50 (He *et al.* 2016)	23M	14M images from ImageNet
VT (Wu *et al.* 2020)	11.7–21.9M	14M images from ImageNet
Wav2vec (Baevski *et al.* 2020)	95–317M	960 hours from LibriSpeech
BERT (Devlin *et al.* 2019)	110–340M	3.3B words from Books/Wiki
ERNIE 2.0 (Sun *et al.* 2020)	110–340M	7.9B en + 15B zh tokens
ERNIE 3.0 (Sun *et al.* 2021)	10B	375B tokens of text, as well as knowledge graph

Dataset	Description
Billion Word Benchmark (Chelba *et al.* 2013)	A billion words of English
Common Crawl (Buck, Heafield, and van Ooyen 2014)	https://github.com/commoncrawl
Book Corpus (Zhu *et al.* 2015)	Speech with text
ImageNet (Deng *et al.* 2009	14M images, annotated with 21k classes
LibriSpeech (Panayotov *et al.* 2015)	960 hours of speech with text
LJ Speech	https://keithito.com/LJ-Speech-Dataset/

What is ~~fine-tuning~~ *fit*?

- *fit*: $f_{pre} + data \rightarrow f_{post}$
- f_{pre}: BERT
  - Trained on Text Corpora
- $data$: SQuAD
  - Maps questions + contexts to answers
- f_{post}:
  - Maps questions + contexts to answers

SQuAD

- Question:
  - *What does AFC stand for*?
- Context:
  - *The American Football Conference (AFC) champion Denver Broncos defeated the National Football Conference (NFC) champion Carolina Panthers 24–10 to earn their third Super Bowl title*.
- Answer:
  - *American Football Conference*

Text with no markup

f_{pre}: Pre-trained Model

Pre-trained (foundation) model	rameters	Training data
ResNet-50 (He *et al.* 2016)	23M	14M images from ImageNet
VT (Wu *et al.* 2020)	11.7–21.9M	14M images from ImageNet
Wav2vec (Baevski *et al.* 2020)	95–317M	960 hours from LibriSpeech
BERT (Devlin *et al.* 2019)	110–340M	3.3B words from Books/Wiki
ERNIE 2.0 (Sun *et al.* 2020)	110–340M	7.9B en + 15B zh tokens
ERNIE 3.0 (Sun *et al.* 2021)	10B	375B tokens of text, as well as knowledge graph

Dataset	Description
Billion Word Benchmark (Chelba *et al.* 2013)	A billion words of English
Common Crawl (Buck, Heafield, and van Ooyen 2014)	https://github.com/commoncrawl
Book Corpus (Zhu *et al.* 2015)	Speech with text
ImageNet (Deng *et al.* 2009)	14M images, annotated with 21k classes
LibriSpeech (Panayotov *et al.* 2015)	960 hours of speech with text
LJ Speech	https://keithito.com/LJ-Speech-Dataset/

gft Design Goals

- Inclusiveness:
  - Accessible to Masses
    - (including non-programmers)
  - ➢**Portability**
  - Ease-of-Use/Transparency >> SOTA
- Hide Complexity/Magic (Alchemy)
  - No one would suggest
    - Regression-like methods are ``intelligent''

Portability:

Inclusiveness + Hedge

gft Programs

H → HuggingFace

```
gft_fit --model H:bert-base-cased \
    --data H:emotion \
    --output_dir $1 \
    --eqn 'classify: label ~ text'
```

Hedge
(Supply Chains)

P → PaddleNLP

```
gft_fit --model P:ernie-tiny \
    --data P:chnsenticorp \
    --output_dir $1 \
    --eqn 'classify: label ~ text'
```

Emphasis on Chinese

Strategy Conclusions

- Inclusiveness
  - Appeal to Power Users as well as Masses
  - Accessible to Masses
  - Portability
- Transparency
  - Avoid Magic / Alchemy
  - Hide Complexity
- Embrace Historical Precedents
  - Little Languages from Unix
  - Fit/Predict/Summary/glm Equations from R

Agenda: Part A

✓Strategy

➢ *gft* Cheatsheet
- 4-5 Functions + 4 Arguments
- Amazing how much can be done with so little

- Seven Simple Examples

- More Tasks: MT, ASR, Vision
- Two Questions
  - How do we *find* the good stuff?
  - How do we *use* the good stuff?
- Why de-emphasize pre-training?
- Conclusions

gft Cheat Sheet
(General Fine-Tuning)

4+1 Functions

1. gft_fit: $f_{pre} \rightarrow f_{post}$ (fine-tuning)
   - 4 Arguments, --output_dir, --metric, --splits
   - (plus most args in most hubs)
2. gft_predict: $f(x) \rightarrow \hat{y}$ (inference)
   - Input: 4 Arguments (x from data or stdin)
   - Output: $\hat{y}$ for each x
3. gft_eval: Score model on dataset
   - Input: 4 Arguments, --split, --metric, …
   - Output: Score
4. gft_summary: Find good stuff
   - Input: 4 Arguments
   - (may include: __*contains*__, __*infer*__)
5. gft_cat_dataset: Output data to *stdout*
   - Input: 4 Arguments (--data, --eqn)

4 Arguments

Next topic

- **--data**
- --model
- --eqn $task: y \sim x_1 + x_2$
- --task
   1. classify (text-classification)
   2. classify_tokens (token-classification)
   3. classify_spans (QA, question-answering)
   4. classify_audio (audio-classification)
   5. classify_images (image-classification)
   6. regress
   7. text-generation
   8. MT (translation)
   9. ASR (ctc, automatic-speech-recognition)
   10. fill-mask

Hugging Face Search models, datasets, users...

Emotion Dataset

$classify: label \sim text$

Datasets: **emotion** like 18

Tasks: multi-class-classification text-classification-other-emotion-classification Task Categories: text-classification

Language Creators: machine-generated Annotations Creators: machine-generated Source Datasets: original

Dataset card Files and versions

Task

Columns

Dataset Preview

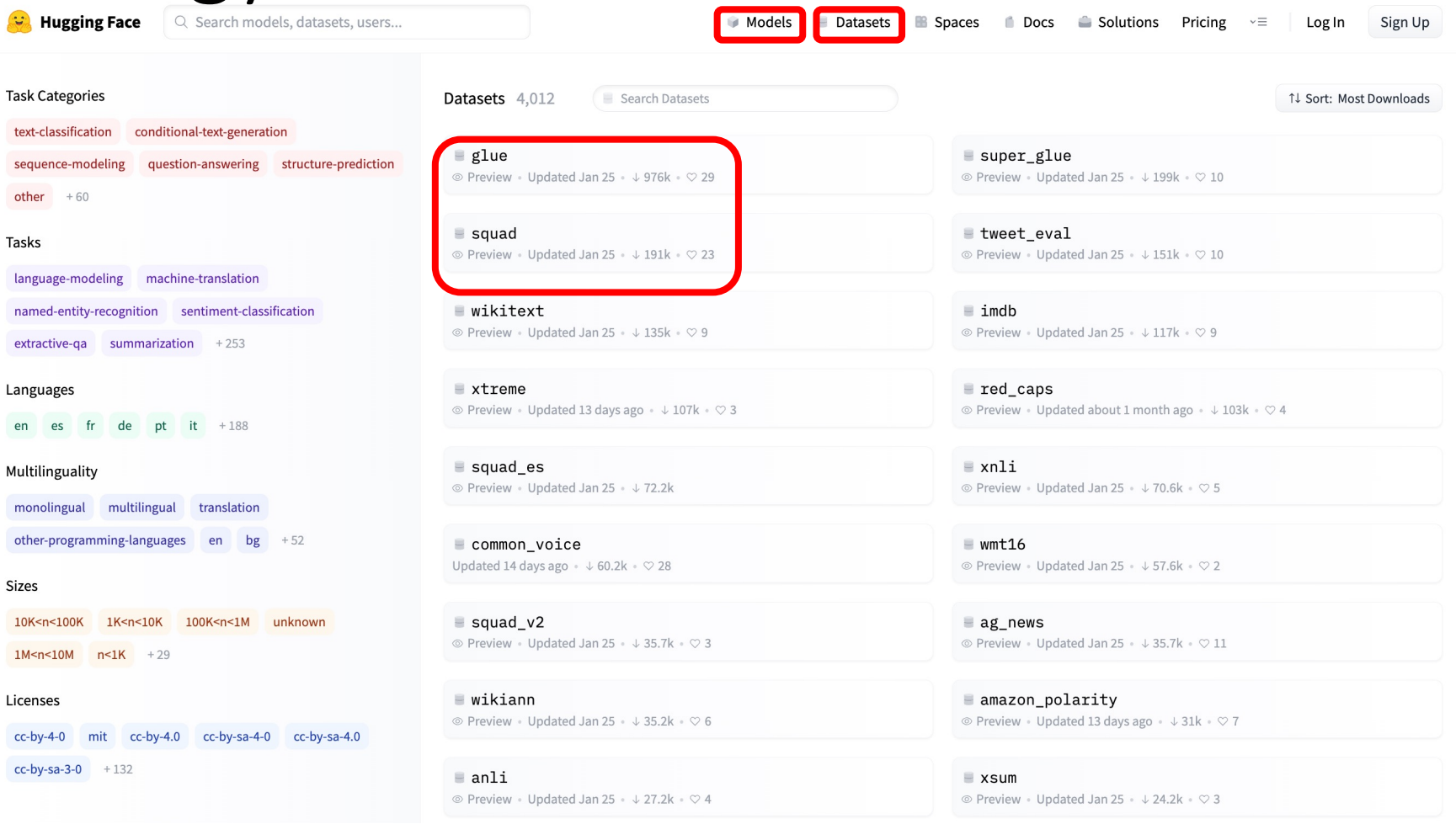
Subset: default Split: train

text (string)	label (class label)
i didnt feel humiliated	0 (sadness)
i can go from feeling so hopeless to so damned hopeful just from being around someone who cares and is awake	0 (sadness)
im grabbing a minute to post i feel greedy wrong	3 (anger)
i am ever feeling nostalgic about the fireplace i will know that it is still on the property	2 (love)
i am feeling grouchy	3 (anger)
ive been feeling a little burdened lately wasnt sure why that was	0 (sadness)
ive been taking or milligrams or times recommended amount and ive fallen asleep a lot faster but i also feel like so funny	5 (surprise)
i feel as confused about life as a teenager or as jaded as a year old man	4 (fear)
i have been with petronas for years i feel that petronas has performed well and made a huge profit	1 (joy)
i feel romantic too	2 (love)
i feel like i have to make the suffering i m seeing mean something	0 (sadness)

Row

More Terminology

- Dataset (4k)
- Models (40k)
- Tasks
  - classify
  - regress
  - translation
  - ASR
  - …



https://huggingface.co/datasets

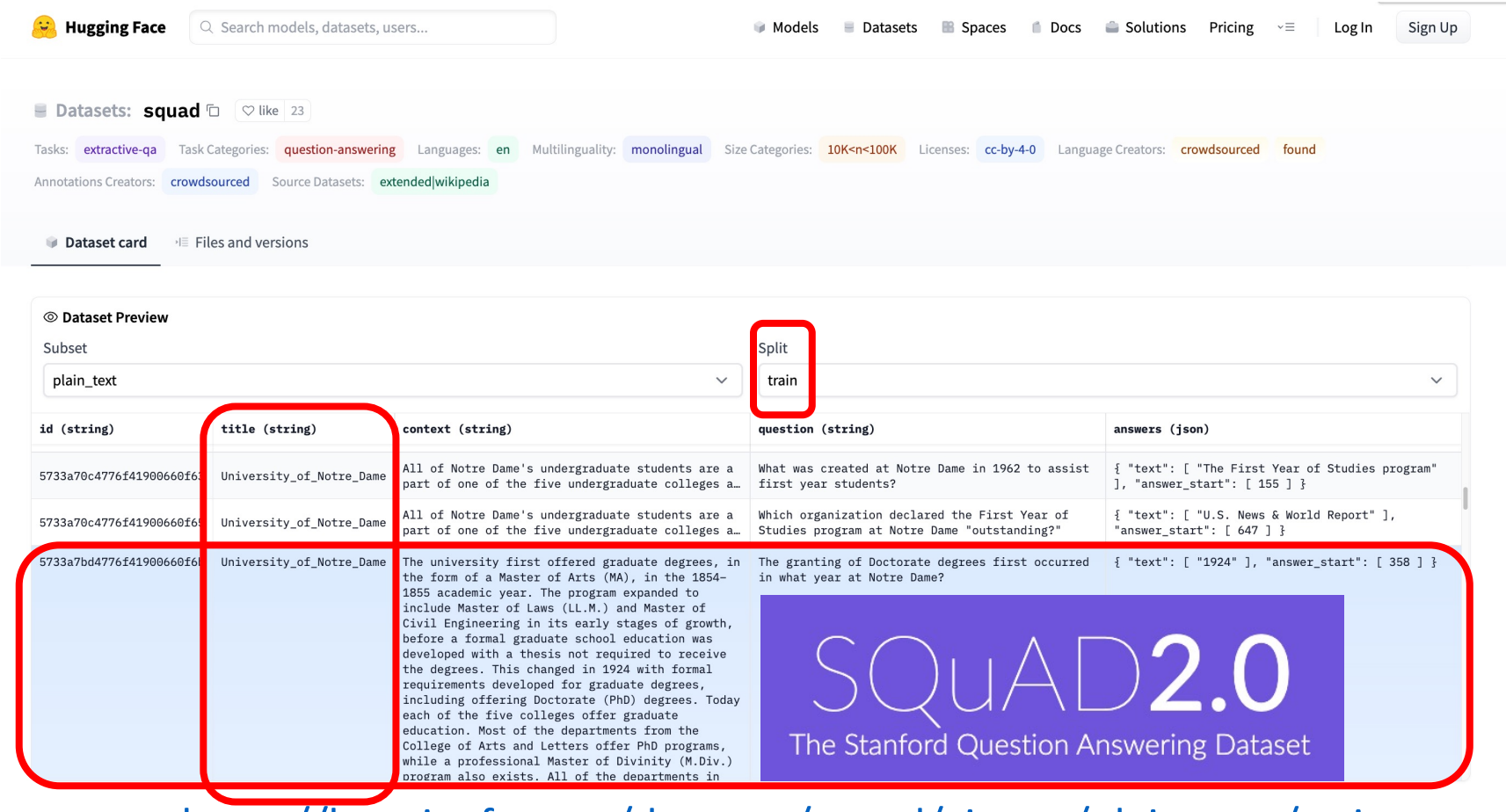
Two Popular Datasets

https://gluebenchmark.com/

https://rajpurkar.github.io/SQuAD-explorer/

More Terminology

- Dataset (4k)
  - Splits: train, validation, test
  - Columns:
    - id, title, context, question, answers
  - Rows
- Models (36k)
- Tasks:
  - classification, regression, etc.



https://huggingface.co/datasets/squad/viewer/plain_text/train

Dataset Preview

Subset: plain_text

Split: train

id (string)	title (string)	context (string)	question (string)	answers (json)
5733a70c4776f41900660f63	University_of_Notre_Dame	All of Notre Dame's undergraduate students are a part of one of the five undergraduate colleges a…	What was created at Notre Dame in 1962 to assist first year students?	{ "text": ["The First Year of Studies program"], "answer_start": [155] }
5733a70c4776f41900660f65	University_of_Notre_Dame	All of Notre Dame's undergraduate students are a part of one of the five undergraduate colleges a…	Which organization declared the First Year of Studies program at Notre Dame "outstanding?"	{ "text": ["U.S. News & World Report"], "answer_start": [647] }
5733a7bd4776f41900660f6b	University_of_Notre_Dame	The university first offered graduate degrees, in the form of a Master of Arts (MA), in the 1854–1855 academic year. The program expanded to include Master of Laws (LL.M.) and Master of Civil Engineering in its early stages of growth, before a formal graduate school education was developed with a thesis not required to receive the degrees. This changed in 1924 with formal requirements developed for graduate degrees, including offering Doctorate (PhD) degrees. Today each of the five colleges offer graduate education. Most of the departments from the College of Arts and Letters offer PhD programs, while a professional Master of Divinity (M.Div.) program also exists. All of the departments in	The granting of Doctorate degrees first occurred in what year at Notre Dame?	{ "text": ["1924"], "answer_start": [358] }

SQuAD2.0
The Stanford Question Answering Dataset

Answer starts at context[358:]

- Dataset (4k)
  - Splits: train, validation, test
  - Columns:
    - title, context, question, answers
  - Rows

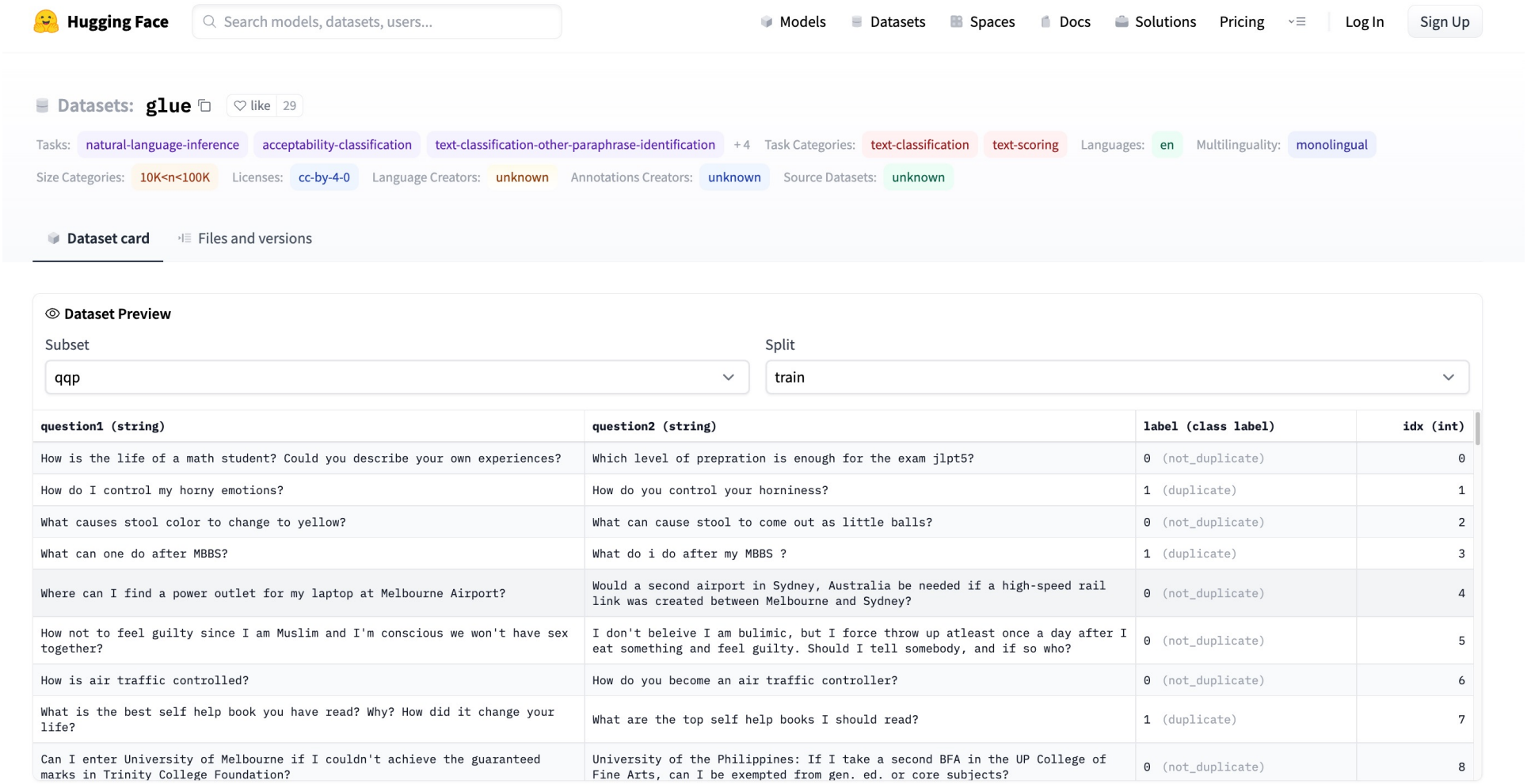
Answer is a span (substring of context)

Task: ~~Question Answering (QA)~~

Classify Spans:
Answer is a substring of context

GLUE

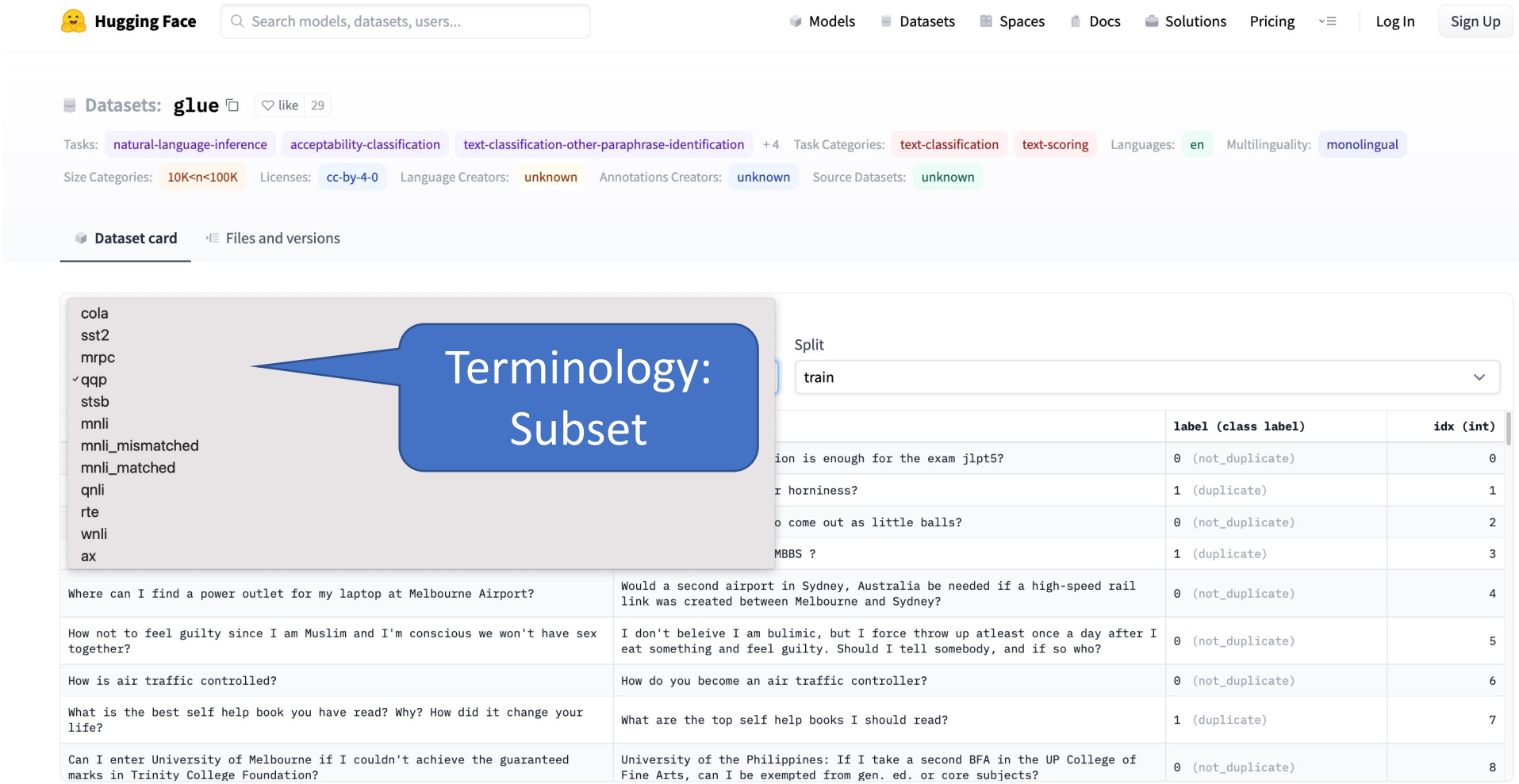
https://huggingface.co/datasets/glue/viewer/qqp/train



question1 (string)	question2 (string)	label (class label)	idx (int)
How is the life of a math student? Could you describe your own experiences?	Which level of prepration is enough for the exam jlpt5?	0 (not_duplicate)	0
How do I control my horny emotions?	How do you control your horniness?	1 (duplicate)	1
What causes stool color to change to yellow?	What can cause stool to come out as little balls?	0 (not_duplicate)	2
What can one do after MBBS?	What do i do after my MBBS ?	1 (duplicate)	3
Where can I find a power outlet for my laptop at Melbourne Airport?	Would a second airport in Sydney, Australia be needed if a high-speed rail link was created between Melbourne and Sydney?	0 (not_duplicate)	4
How not to feel guilty since I am Muslim and I'm conscious we won't have sex together?	I don't beleive I am bulimic, but I force throw up atleast once a day after I eat something and feel guilty. Should I tell somebody, and if so who?	0 (not_duplicate)	5
How is air traffic controlled?	How do you become an air traffic controller?	0 (not_duplicate)	6
What is the best self help book you have read? Why? How did it change your life?	What are the top self help books I should read?	1 (duplicate)	7
Can I enter University of Melbourne if I couldn't achieve the guaranteed marks in Trinity College Foundation?	University of the Philippines: If I take a second BFA in the UP College of Fine Arts, can I be exempted from gen. ed. or core subjects?	0 (not_duplicate)	8

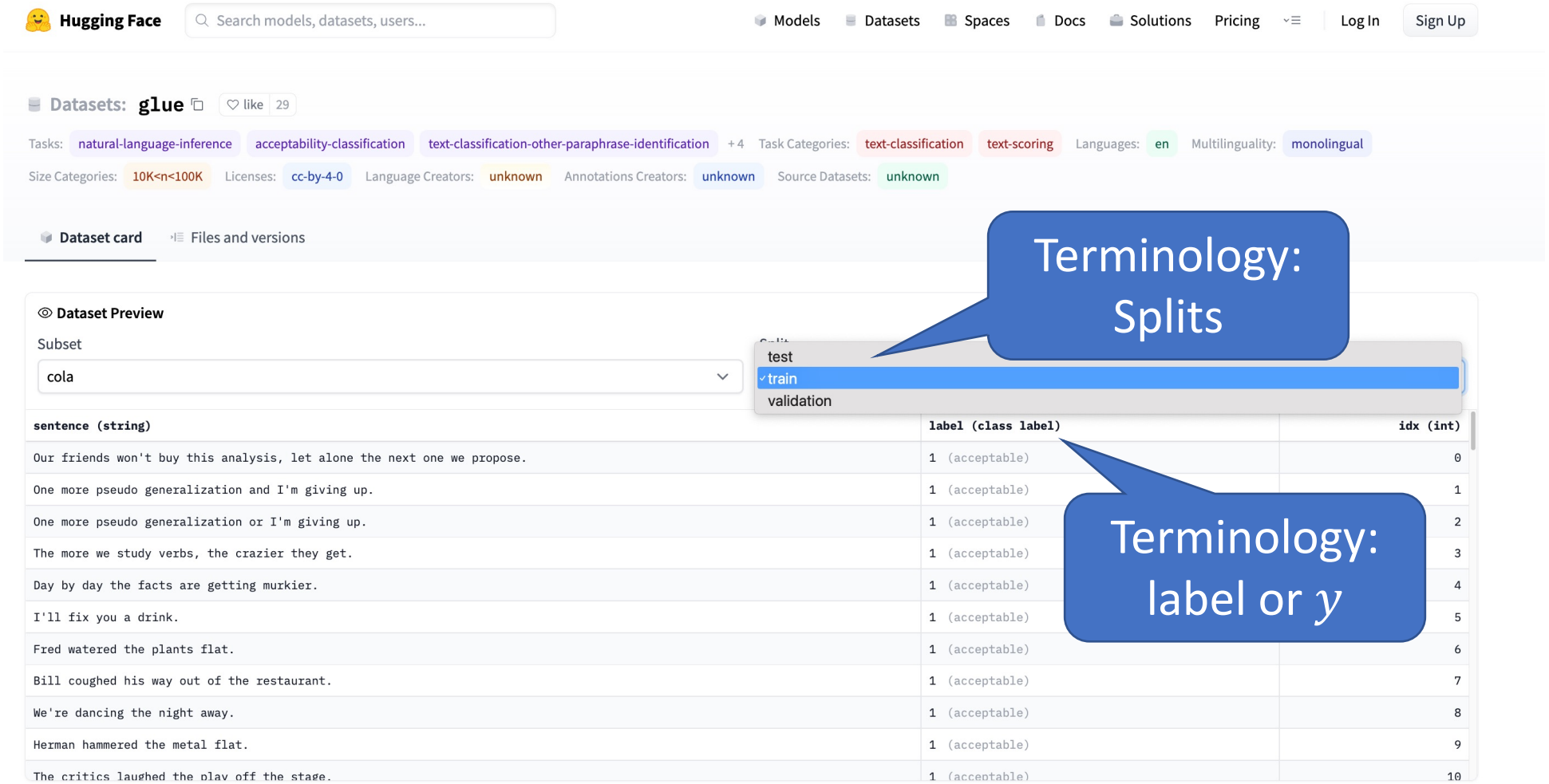
GLUE

https://huggingface.co/datasets/glue/viewer/qqp/train



GLUE

https://huggingface.co/datasets/glue/viewer/qqp/train



Two Perspectives on Datasets

Machine Learning

- Splits: train/val/test
- Treat rows as iid
  - Identical and independently distributed
- Idealistic, but maybe unrealistic
  - Changes over time, Context, etc.
- More papers on models
  - than Datasets
- Relatively little interest in
  - Is the Dataset Realistic?
  - Motivation? (Does anyone care? If so, who?)
  - What is the dataset testing?
  - Sampling/balance? From which population?
  - Can it be gamed? Leakage?

Psychology (Presupposes ∃ Hypothesis)

- Validity
  - Content validity
  - Construct validity
  - Criterion validity
  - Face validity
  - Discriminant validity
- Reliability
  - Inter-rater reliability
  - Test-retest reliability
  - Internal reliability/Inter-item consistency
  - Split half reliability

Lexicography

Discussion of Validity Assumes: Intention

TYPES OF VALIDITY

Experimental Validity: is the study really measuring what it intends?

INTERNAL VALIDITY refers to things that happen "inside" the study. Internal validity is concerned with whether we can be certain that it was the IV which caused the change in the DV. If aspects of the experimental situation lack validity, the results of the study are meaningless and we can make no meaningful conclusions from them.

- Internal validity can be affected by a lack of **mundane realism**. This could lead the participants to act in a way which is unnatural, thus making the results less valid.
- Internal validity can also be affected by **extraneous variables** (see below).

EXTRANEOUS VARIABLE	HOW DOES IT AFFECT VALIDITY?	HOW CAN IT BE OVERCOME?
Situational variables (anything to do with the environment of the experiment): time of day, temperature, noise levels etc	Something about the situation of the experiment could act as an EV if it has an effect on the DV. For example, poor lighting could affect participants performance on a memory test	Situational variables can be overcome by the use of **standardised procedures** which ensure that all participants are tested under the same conditions.
Participants variables (anything to do with differences in the participants): age, gender, intelligence, skill, past experience, motivation, education etc.	It may be that the differences between the participants cause the change in the DV. For example, one group may perform better on a memory test than another because they are younger, or more motivated.	Participant variables can be completely removed by using a **repeated measures design** (the same participants are used in each condition). **Matched pairs** (participants in each group are matched) could also be used.
Investigator effects: this refers to how the behaviour and language of the experimenter may influence the behaviour of the participants. The way in which an experimenter asks a question might act as a cue for the participant. Also known as **experimenter bias**	**Leading questions** from the experimenter may consciously or unconsciously alter how the participant responds. For example, the experimenter may provide verbal or non verbal encouragement when the participant behaves in a way which supports the hypothesis.	Investigator effects can be overcome by using a **double blind** technique. This is when the person who carries out the research is not the person who designed it.
Demand characteristics: participants are often searching for cues as to how to behave in an experiment. There could be something about the experimental situation or the behaviour of the experimenter (see **investigator effects**) which communicates to the participant what is "demanded" of them.	The structure of the experiment could lead the participant to guess the aim of the study. For example, participants may perform a memory test, be made to exercise, and then given another memory test. This may lead the participants to guess that the study is about the effect of exercise on memory, which may cause them to change their behaviour	When designing a study, it is important to try and create a situation where the participants will not be able to guess what the aim of the study is.
Participant effects: participants are aware that they are in an experiment, and so may behave unnaturally.	They may be overly helpful and want to please the experimenter. This leads to artificial behaviour. Alternatively, they may decide to go against the experimenter's aims and deliberately act in a way which spoils the experiment. This is the "**screw you**" effect.	Again, by designing a study so that the participants cannot guess the aims, participant effects can be reduced.

gft Cheat Sheet
(General Fine-Tuning)

4+1 Functions

1. gft_fit: $f_{pre} \rightarrow f_{post}$ (fine-tuning)
   - 4 Arguments, --output_dir, --metric, --splits
   - (plus most args in most hubs)
2. gft_predict: $f(x) \rightarrow \hat{y}$ (inference)
   - Input: 4 Arguments (x from data or stdin)
   - Output: $\hat{y}$ for each x
3. gft_eval: Score model on dataset
   - Input: 4 Arguments, --split, --metric, …
   - Output: Score
4. gft_summary: Find good stuff
   - Input: 4 Arguments
   - (may include: *__contains__*, *__infer__*)
5. gft_cat_dataset: Output data to *stdout*
   - Input: 4 Arguments (--data, --eqn)

4 Arguments

✓ --data

➢ **--model**

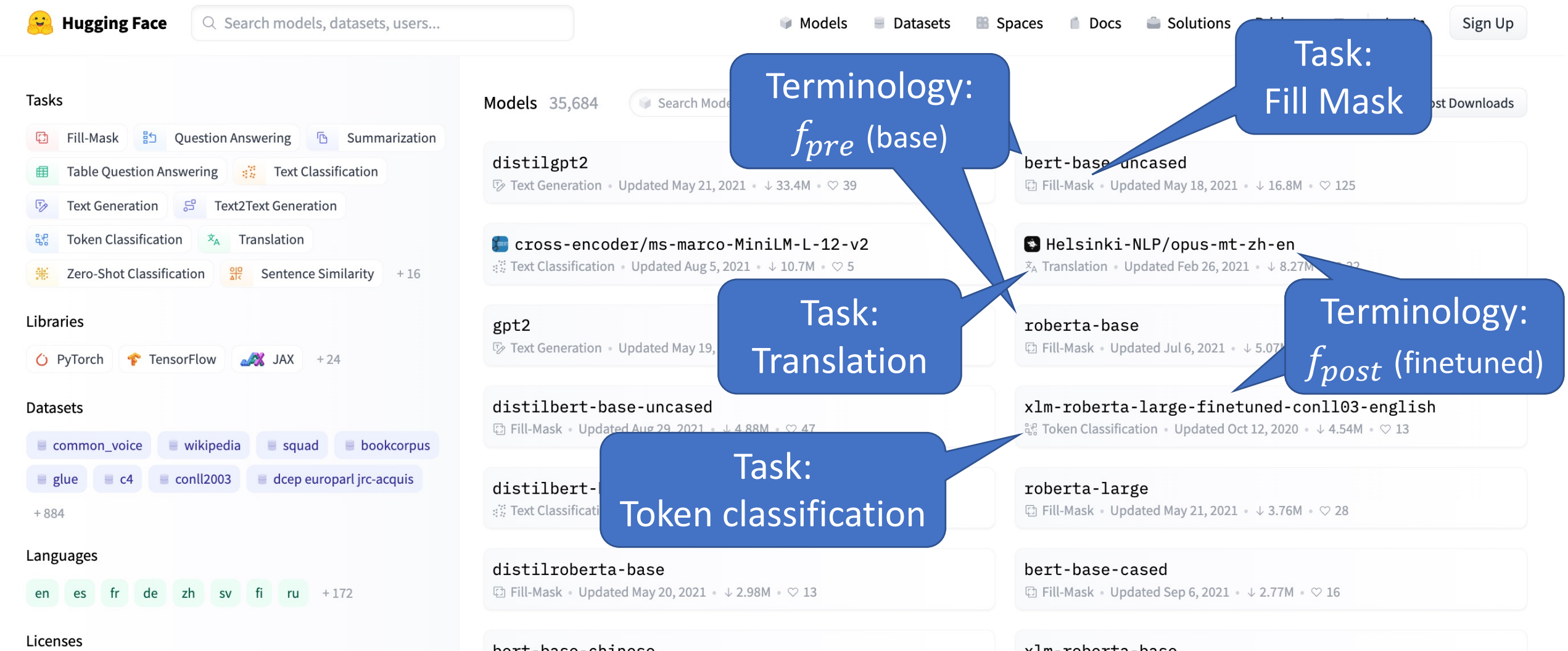
Next topic

- --eqn $task: y \sim x_1 + x_2$
- --task
  1. classify (text-classification)
  2. classify_tokens (token-classification)
  3. classify_spans (QA, question-answering)
  4. classify_audio (audio-classification)
  5. classify_images (image-classification)
  6. regress
  7. text-generation
  8. MT (translation)
  9. ASR (ctc, automatic-speech-recognition)
  10. fill-mask

Models

Terminology: f, f_{pre}, f_{post}

https://huggingface.co/models



gft Cheat Sheet
(General Fine-Tuning)

4+1 Functions

1. gft_fit: $f_{pre} \rightarrow f_{post}$ (fine-tuning)
   - 4 Arguments, --output_dir, --metric, --splits
   - (plus most args in most hubs)
2. gft_predict: $f(x) \rightarrow \hat{y}$ (inference)
   - Input: 4 Arguments (x from data or stdin)
   - Output: $\hat{y}$ for each x
3. gft_eval: Score model on dataset
   - Input: 4 Arguments, --split, --metric, …
   - Output: Score
4. gft_summary: Find good stuff
   - Input: 4 Arguments
   - (may include: *__contains__*, *__infer__*)
5. gft_cat_dataset: Output data to *stdout*
   - Input: 4 Arguments (--data, --eqn)

4 Arguments

✓ --data

- --model

➢ **--eqn** $\boldsymbol{task: y \sim x_1 + x_2}$

- --task
  1. classify (text-classification)
  2. classify_tokens (token-classification)
  3. classify_spans (QA, question-answering)
  4. classify_audio (audio-classification)
  5. classify_images (image-classification)
  6. regress
  7. text-generation
  8. MT (translation)
  9. ASR (ctc, automatic-speech-recognition)
  10. fill-mask

Next topic

Simple Equations Cover Many Cases of Interest
GLUE: A Popular Benchmark

Subset	Dataset	Equation
COLA	H:glue,cola	$classify: label \sim sentence$
SST2	H:glue,sst2	$classify: label \sim sentence$
WNLI	H:glue,wnli	$classify: label \sim sentence$
MRPC	H:glue,mrpc	$classify: label \sim sentence1 + sentence2$
QNLI	H:glue,qnli	$classify: label \sim sentence1 + sentence2$
QQP	H:glue,qqp	$classify: label \sim question + sentence$
SSTB	H:glue,sstb	$regress: label \sim question1 + question2$
MNLI	H:glue,mnli	$classify: label \sim premise + hypothesis$

Loss Functions and Metrics (Defaults depend on task)

- --eqn
  - $classify: \ label \sim question + sentence$
  - $regress: \ label \sim question1 + question2$
- Loss functions (used in *gft_fit*)
  - Classification → Cross entropy
  - Regression → Mean squared error
- Metrics (used in *gft_eval*)
  - Classification → Accuracy
  - Regression → Mean squared error
  - --metric H:glue,cola → Matthews Correlation

Concern: Loss ≠ Metric

Equation Keywords ≈ Pipeline Tasks

Benchmark	Subset	Dataset	Equation
GLUE	COLA	H:glue,cola	$classify: label \sim sentence$
SQuAD 1.0		H:squad	$classify\_spans: answers \sim question + context$
SQuAD 2.0		H:squad_v2	$classify\_spans: answers \sim question + context$
CONLL2003	POS	H:conll2003	$classify\_tokens: pos\_tags \sim tokens$
	NER	H:conll2003	$classify\_tokens: ner\_tags \sim tokens$
TIMIT		H:timit_asr	$ctc: text \sim audio$
Amazon Reviews		H:amazon_reviews_multi	$classify: label \sim question + sentence$
VAD		C:$gft/datasets/VAD/VAD	$regress: Valence + Arousal + Dominance \sim Word$

$\mathbb{R}^3$

$regress: Valence + Arousal + Dominance \sim Word$

https://saifmohammad.com/WebPages/nrc-vad.html

word	Val	Arousal	Dom	Dist
open	0.620	0.480	0.569	0.00
unfold	0.612	0.510	0.520	0.06
reopen	0.656	0.528	0.568	0.06
close	0.292	0.260	0.263	0.50
closed	0.240	0.164	0.318	0.55
undecided	0.286	0.433	0.127	0.56

Table 12: Words above the double line are near *open*. The last column is the Euclidean distance to *open*.

	Antonyms				Sim
	adj	noun	verb	fallows	SimLex
cor	0.55	0.48	0.44	0.52	-0.40

Table 13: VAD distances are positively correlated with antonyms, and negatively correlated with SimLex similarities, though none of these correlations are large.

$\mathbb{R}^3$

$regress: Valence + Arousal + Dominance \sim Word$

https://saifmohammad.com/WebPages/nrc-vad.html

word	Val	Arousal	Dom	Dist
open	0.620	0.480	0.569	0.00
unfold	0.612	0.510	0.520	0.06
reopen	0.656	0.528	0.568	0.06
close	0.292	0.260	0.263	0.50
closed	0.240	0.164	0.318	0.55
undecided	0.286	0.433	0.127	0.56

Table 12: Words above the double line are near *open*. The last column is the Euclidean distance to *open*.

Train f on words, but apply f to texts
(Generalizations: OOVs, MWEs, negation)

```
gft_fit --model H:bert-base-cased \
        --data  C:$gft/datasets/VAD/VAD \
        --eqn   'regress: Valence + Arousal + Dominance ~ Word'
        --output_dir $outdir
```

Listing 20. An equation with a vector on the left hand side (lhs)

Input, x	Predictions, $\hat{y}$ $\hat{V}$	$\hat{A}$	$\hat{D}$	Gold, y V	A	D
love	0.976	0.530	0.675	1.000	0.519	0.673
she loves me	0.931	0.452	0.648	NA	NA	NA
lovable	0.920	0.318	0.608	0.948	0.335	0.565
she loves me not	0.382	0.375	0.349	NA	NA	NA
ugly duckling	0.300	0.517	0.242	NA	NA	NA
unlovable	0.203	0.502	0.399	NA	NA	NA
she does not love me	0.189	0.433	0.262	NA	NA	NA
she hates me	0.135	0.685	0.363	NA	NA	NA
loath	0.094	0.790	0.413	0.135	0.714	0.445
hate	0.017	0.780	0.451	0.031	0.802	0.430

Table 7. *Some gold labels and predictions from model, f_{post}, from Listing 20*

gft Cheat Sheet
(General Fine-Tuning)

4+1 Functions

1. gft_fit: $f_{pre} \rightarrow f_{post}$ (fine-tuning)
   - 4 Arguments, --output_dir, --metric, --splits
   - (plus most args in most hubs)
2. gft_predict: $f(x) \rightarrow \hat{y}$ (inference)
   - Input: 4 Arguments (x from data or stdin)
   - Output: $\hat{y}$ for each x
3. gft_eval: Score model on dataset
   - Input: 4 Arguments, --split, --metric, …
   - Output: Score
4. gft_summary: Find good stuff
   - Input: 4 Arguments
   - (may include: __*contains*__, __*infer*__)
5. gft_cat_dataset: Output data to *stdout*
   - Input: 4 Arguments (--data, --eqn)

4 Arguments

✓ --data

✓ --model

✓ --eqn $task: y \sim x_1 + x_2$

➢ **--task**

1. classify (text-classification)
2. classify_tokens (token-classification)
3. classify_spans (QA, question-answering)
4. classify_audio (audio-classification)
5. classify_images (image-classification)
6. regress
7. text-generation
8. MT (translation)
9. ASR (ctc, automatic-speech-recognition)
10. fill-mask

Tasks

- classify, text-classification — $y \in \{0,1,2, \ldots\}$
- regress — $y \in \mathbb{R}$ or $y \in \mathbb{R}^N$
- QA, Question Answering, classify spans — y for each start/end of span
- token classification — y for each token
  - NER (Named Entity Recognition)
  - POS (Part of Speech Tagging)
- translation, MT
- ASR, Automatic Speech Recognition, ctc — y for each phoneme

gft Cheat Sheet
(General Fine-Tuning)

4+1 Functions

1. gft_fit: $f_{pre} \rightarrow f_{post}$ (fine-tuning)
   - 4 Arguments, --output_dir, --metric, --splits
   - (plus most args in most hubs)
2. gft_predict: $f(x) \rightarrow \hat{y}$ (inference)
   - Input: 4 Arguments (x from data or stdin)
   - Output: $\hat{y}$ for each x
3. gft_eval: Score model on dataset
   - Input: 4 Arguments, --split, --metric, …
   - Output: Score
4. gft_summary: Find good stuff
   - Input: 4 Arguments
   - (may include: __*contains*__, __*infer*__)
5. gft_cat_dataset: Output data to *stdout*
   - Input: 4 Arguments (--data, --eqn)

4 Arguments

✓ --data

✓ --model

✓ --eqn $task: y \sim x_1 + x_2$

✓ **--task**

1. classify (text-classification)
2. classify_tokens (token-classification)
3. classify_spans (QA, question-answering)
4. classify_audio (audio-classification)
5. classify_images (image-classification)
6. regress
7. text-generation
8. MT (translation)
9. ASR (ctc, automatic-speech-recognition)
10. fill-mask

Agenda: Part A

✓Strategy

✓*gft* Cheatsheet

- 4-5 Functions + 4 Arguments
- Amazing how much can be done with so little

➢**Seven Simple Examples**

- More Tasks: MT, ASR, Vision
- Two Questions
  - How do we *find* the good stuff?
  - How do we *use* the good stuff?
- Why de-emphasize pre-training?
- Conclusions

Seven Simple Examples

1. Summary
2. Summary with patterns (__*contains*__)
3. Predict: Sentiment Analysis & Text Classification
4. More tasks: token_classification, fill-mask
5. Input from datasets (as opposed to stdin)
6. *gft_predict* → *gft_eval*
7. *gft_fit* (aka fine-tuning)

Seven Simple Examples: 1 of 7
Summarize (almost) anything

```
emodel=H:bhadresh-savani/roberta-base-emotion

# Summarize a dataset and/or model
gft_summary --data H:emotion
gft_summary --model $emodel
gft_summary --data H:emotion --model $emodel
```

Emotion Classes

Summarize both data and model

Seven Simple Examples: 2 of 7

Summary with patterns (__*contains*__)

```
emodel=H:bhadresh-savani/roberta-base-emotion

# Summarize a dataset and/or model
gft_summary --data H:emotion
gft_summary --model $emodel
gft_summary --data H:emotion --model $emodel

# find some popular datasets and models that contain "emotion"
gft_summary --data H:__contains__emotion --topn 5
gft_summary --model H:__contains__emotion --topn 5
```

Search Engine
(Embarrassment of Riches)

Find datasets on HuggingFace that contain “emotion”

Find *models* on HuggingFace that contain “emotion”

Seven Simple Examples: 3 of 7

Predict: Sentiment Analysis & Text Classification

```
emodel=H:bhadresh-savani/roberta-base-emotion

# Summarize a dataset and/or model
gft_summary --data H:emotion
gft_summary --model $emodel
gft_summary --data H:emotion --model $emodel

# find some popular datasets and models that contain "emotion"
gft_summary --data H:__contains__emotion --topn 5
gft_summary --model H:__contains__emotion --topn 5

# make predictions on inputs from stdin
echo 'I love you.' | gft_predict --task classify

# The default model (for the classification task) performs sentiment analysis
# The model, $emodel, outputs emotion classes (as opposed to POSITIVE/NEGATIVE)
echo 'I love you.' | gft_predict --task classify --model $emodel
```

Default model → Sentiment Analysis

Emotion Classes

Seven Simple Examples: 4 of 7

More tasks: token_classification, fill-mask

```
emodel=H:bhadresh-savani/roberta-base-emotion

# Summarize a dataset and/or model
gft_summary --data H:emotion
gft_summary --model $emodel
gft_summary --data H:emotion --model $emodel

# find some popular datasets and models that contain "emotion"
gft_summary --data H:__contains__emotion --topn 5
gft_summary --model H:__contains__emotion --topn 5

# make predictions on inputs from stdin
echo 'I love you.' | gft_predict --task classify
```

```
# The default model (for the classification task) performs sentiment analysis
# The model, $emodel, outputs emotion classes (as opposed to POSITIVE/NEGATIVE)
echo 'I love you.' | gft_predict --task classify --model $emodel

# some other tasks (beyond classification)
echo 'I love New York.' | gft_predict --task H:token-classification
echo 'I <mask> you.' | gft_predict --task H:fill-mask
```

~~NER: Named Entity Recognition~~

Cloze (1953)

https://en.wikipedia.org/wiki/Cloze_test

Seven Simple Examples: 5 of 7

Input from datasets (as opposed to stdin)

```
# some other tasks (beyond classification)
echo 'I love New York.' | gft_predict --task H:token-classification
echo 'I <mask> you.' | gft_predict --task H:fill-mask

# make predictions on inputs from a split of a standard dataset
gft_predict --eqn 'classify: label ~ text' --model $emodel --data H:emotion --split test
```

Specify split

Equation specifies task & columns

Read Input from dataset (as opposed to stdin)

Seven Simple Examples: 6 of 7
gft_predict → gft_eval

```
# some other tasks (beyond classification)
echo 'I love New York.' | gft_predict --task H:token-classification
echo 'I <mask> you.' | gft_predict --task H:fill-mask

# make predictions on inputs from a split of a standard dataset
gft_predict --eqn 'classify: label ~ text' --model $emodel --data H:emotion --split test

# return a single score (as opposed to a prediction for each input)
gft_eval --eqn 'classify: label ~ text' --model $emodel --data H:emotion --split test
```

Eval returns a single score for the dataset split
(as opposed to a prediction for each example in dataset split)

A Few Simple Examples: 7 of 7
gft_fit (aka fine-tuning)

```
# some other tasks (beyond classification)
echo 'I love New York.' | gft_predict --task H:token-classification
echo 'I <mask> you.' | gft_predict --task H:fill-mask

# make predictions on inputs from a split of a standard dataset
gft_predict --eqn 'classify: label ~ text' --model $emodel --data H:emotion --split test

# return a single score (as opposed to a prediction for each input)
gft_eval --eqn 'classify: label ~ text' --model $emodel --data H:emotion --split test

# Input a pre-trained model (bert) and output a post-trained model
gft_fit --eqn 'classify: label ~ text' \
    --model H:bert-base-cased \
    --data H:emotion \
    --output_dir $outdir
```

Fit returns a post-trained model: f_{post}

Recap of Seven Simple Examples

1

```
# Summarize a dataset and/or model
gft_summary --data H:emotion
gft_summary --model $emodel
gft_summary --data H:emotion --model $emodel

# find some popular datasets and models that contain "emotion"
gft_summary --data H:__contains__emotion --topn 5
gft_summary --model H:__contains__emotion --topn 5

# make predictions on inputs from stdin
echo 'I love you.' | gft_predict --task classify

# The default model (for the classification task) performs sentiment analysis
# The model, $emodel, outputs emotion classes (as opposed to POSITIVE/NEGATIVE)
echo 'I love you.' | gft_predict --task classify --model $emodel
```

2

3

4

```
# some other tasks (beyond classification)
echo 'I love New York.' | gft_predict --task H:token-classification
echo 'I <mask> you.' | gft_predict --task H:fill-mask
```

5

```
# make predictions on inputs from a split of a standard dataset
gft_predict --eqn 'classify: label ~ text' --model $emodel --data H:emotion --split test
```

6

```
# return a single score (as opposed to a prediction for each input)
gft_eval --eqn 'classify: label ~ text' --model $emodel --data H:emotion --split test
```

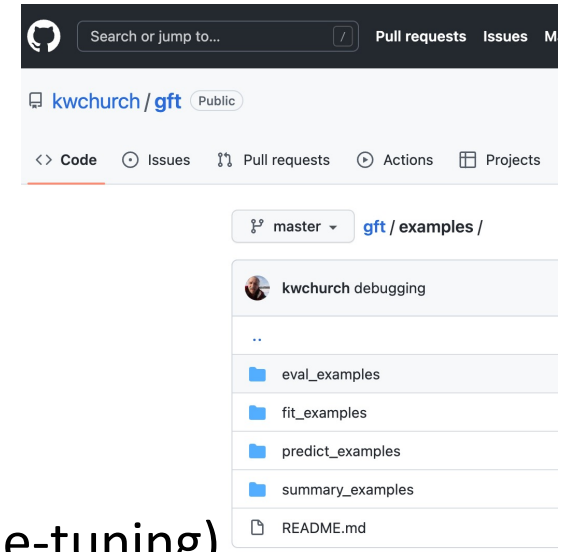
7

```
# Input a pre-trained model (bert) and output a post-trained model
gft_fit --eqn 'classify: label ~ text' \
    --model H:bert-base-cased \
    --data H:emotion \
    --output_dir $outdir
```

1. Summary
2. Summary with patterns (__*contains*__)
3. Sentiment Analysis & Text Classification
4. More tasks: token_classification, fill-mask
5. Input from datasets (as opposed to stdin)
6. *gft_predict → gft_eval*
7. *gft_fit* (aka fine-tuning)

Hundreds of Examples:

https://github.com/kwchurch/gft



Subdirectories under $gft/examples

1. fit_examples
2. predict_examples
3. eval_examples
4. summary_examples

- Look for *.sh files under these directories
  - `find $gft/examples -name "*.sh"`
- 4 arguments:
  - --data, --model, --eqn, --task

4 Functions

1. gft_fit: $f_{pre} \rightarrow f_{post}$ (fine-tuning)
   - 4 Arguments, --output_dir, --metric, --splits
   - (plus most args in most hubs)
2. gft_predict: $f(x) \rightarrow \hat{y}$ (inference)
   - Input: 4 Arguments (x from data or stdin)
   - Output: $\hat{y}$ for each x
3. gft_eval: Score model on dataset
   - Input: 4 Arguments, --metric, …
   - Output: Score
4. gft_summary: Find good stuff
   - 4 Arguments
   - (may include: *__contains__*, *__infer__*)

Hundreds of Examples: Regression Testing

Regression testing looks for errors in output

Pass: No Errors

Input

master gft / examples / summary_examples / model / model.HuggingFace / **roberta-base.sh**

kwchurch first commit

1 contributor

Executable File | 3 lines (2 sloc) | 46 Bytes

```
1  #!/bin/sh
2
3  gft_summary --model H:roberta-base
```

Output

master gft / examples / summary_examples / model / model.HuggingFace / **roberta-base.out**

kwchurch debugging

1 contributor

298 lines (298 sloc) | 13.5 KB

```
1  model: roberta-base     model: roberta-base     downloads: 9098576      likes: 23       task: fill-mask
2  model: roberta-base     labels: LABEL_0, LABEL_1
3  RobertaForMaskedLM(
4    (roberta): RobertaModel(
5      (embeddings): RobertaEmbeddings(
6        (word_embeddings): Embedding(50265, 768, padding_idx=1)
7        (position_embeddings): Embedding(514, 768, padding_idx=1)
8        (token_type_embeddings): Embedding(1, 768)
9        (LayerNorm): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
10       (dropout): Dropout(p=0.1, inplace=False)
11     )
12     (encoder): RobertaEncoder(
13       (layer): ModuleList(
14         (0): RobertaLayer(
15           (attention): RobertaAttention(
16             (self): RobertaSelfAttention(
17               (query): Linear(in_features=768, out_features=768, bias=True)
18               (key): Linear(in_features=768, out_features=768, bias=True)
19               (value): Linear(in_features=768, out_features=768, bias=True)
20               (dropout): Dropout(p=0.1, inplace=False)
```

Guide to Hundreds of Examples

```
(gft-adapters) kwc@asimov-7:~/gft7/gft/examples$ find . -name '*sh' | sed 50q
./eval_examples/model.HuggingFace/language/data.HuggingFace/Yuchen_models/glue/mrpc.sh
./eval_examples/model.HuggingFace/language/data.HuggingFace/Yuchen_models/glue/sst2.sh
./eval_examples/model.HuggingFace/language/data.HuggingFace/Yuchen_models/glue/qnli.sh
./eval_examples/model.HuggingFace/language/data.HuggingFace/Yuchen_models/glue/rte.sh
./eval_examples/model.HuggingFace/language/data.HuggingFace/Yuchen_models/glue/stsb.sh
./eval_examples/model.HuggingFace/language/data.HuggingFace/their_models/paraphrase/paws_labeled_final.sh
./eval_examples/model.HuggingFace/language/data.HuggingFace/their_models/paraphrase/paws_unlabeled_final.sh
./eval_examples/model.HuggingFace/language/data.HuggingFace/their_models/paraphrase/paws_labeled_swap.sh
./eval_examples/model.HuggingFace/language/data.HuggingFace/their_models/text_classification/banking77.sh
./eval_examples/model.HuggingFace/language/data.HuggingFace/their_models/text_classification/snli.sh
./eval_examples/model.HuggingFace/language/data.HuggingFace/their_models/text_classification/ag_news.sh
./eval_examples/model.HuggingFace/language/data.HuggingFace/their_models/text_classification/tweets_hate_speech_detection.sh
./eval_examples/model.HuggingFace/language/data.HuggingFace/their_models/text_classification/emotion.sh
./eval_examples/model.HuggingFace/language/data.HuggingFace/their_models/glue/cola.sh
./eval_examples/model.HuggingFace/language/data.HuggingFace/their_models/glue/mrpc.sh
./eval_examples/model.HuggingFace/language/data.HuggingFace/their_models/glue/sst2.sh
./eval_examples/model.HuggingFace/language/data.HuggingFace/their_models/glue/mnli_mismatched.sh
./eval_examples/model.HuggingFace/language/data.HuggingFace/their_models/glue/qnli.sh
./eval_examples/model.HuggingFace/language/data.HuggingFace/their_models/glue/qqp.sh
./eval_examples/model.HuggingFace/language/data.HuggingFace/their_models/glue/rte.sh
./eval_examples/model.HuggingFace/language/data.HuggingFace/their_models/glue/mnli_matched.sh
./eval_examples/model.HuggingFace/language/data.HuggingFace/their_models/glue/stsb.sh
./eval_examples/model.HuggingFace/language/data.HuggingFace/their_models/AdapterHub/cola.sh
./eval_examples/model.HuggingFace/language/data.HuggingFace/their_models/AdapterHub/anli_r3.sh
./eval_examples/model.HuggingFace/language/data.HuggingFace/their_models/AdapterHub/mrpc.sh
./eval_examples/model.HuggingFace/language/data.HuggingFace/their_models/AdapterHub/scicite.sh
./eval_examples/model.HuggingFace/language/data.HuggingFace/their_models/AdapterHub/sst2.sh
./eval_examples/model.HuggingFace/language/data.HuggingFace/their_models/AdapterHub/duorc_s.sh
./eval_examples/model.HuggingFace/language/data.HuggingFace/their_models/AdapterHub/winogrande.sh
./eval_examples/model.HuggingFace/language/data.HuggingFace/their_models/AdapterHub/art.sh
./eval_examples/model.HuggingFace/language/data.HuggingFace/their_models/AdapterHub/drop.sh
./eval_examples/model.HuggingFace/language/data.HuggingFace/their_models/AdapterHub/conll2000.sh
./eval_examples/model.HuggingFace/language/data.HuggingFace/their_models/AdapterHub/sick.sh
./eval_examples/model.HuggingFace/language/data.HuggingFace/their_models/AdapterHub/comqa.sh
./eval_examples/model.HuggingFace/language/data.HuggingFace/their_models/AdapterHub/ud_pos.sh
./eval_examples/model.HuggingFace/language/data.HuggingFace/their_models/AdapterHub/mit_movie_trivia.sh
./eval_examples/model.HuggingFace/language/data.HuggingFace/their_models/AdapterHub/squad.sh
./eval_examples/model.HuggingFace/language/data.HuggingFace/their_models/AdapterHub/snli.sh
./eval_examples/model.HuggingFace/language/data.HuggingFace/their_models/AdapterHub/wnut_17.sh
./eval_examples/model.HuggingFace/language/data.HuggingFace/their_models/AdapterHub/wikihop.sh
./eval_examples/model.HuggingFace/language/data.HuggingFace/their_models/AdapterHub/copa.sh
./eval_examples/model.HuggingFace/language/data.HuggingFace/their_models/AdapterHub/emo.sh
./eval_examples/model.HuggingFace/language/data.HuggingFace/their_models/AdapterHub/pmb_sem_tagging.sh
./eval_examples/model.HuggingFace/language/data.HuggingFace/their_models/AdapterHub/boolq.sh
./eval_examples/model.HuggingFace/language/data.HuggingFace/their_models/AdapterHub/rotten_tomatoes.sh
./eval_examples/model.HuggingFace/language/data.HuggingFace/their_models/AdapterHub/quoref.sh
./eval_examples/model.HuggingFace/language/data.HuggingFace/their_models/AdapterHub/trec.sh
./eval_examples/model.HuggingFace/language/data.HuggingFace/their_models/AdapterHub/conll2003_pos.sh
./eval_examples/model.HuggingFace/language/data.HuggingFace/their_models/AdapterHub/wic.sh
./eval_examples/model.HuggingFace/language/data.HuggingFace/their_models/AdapterHub/hotpotqa.sh
```

Naming Conventions

- Function
  - fit, predict, eval, summary
- Models
  - model.HuggingFace/model.Paddle
  - our_models/their_models
- Data
  - data.HuggingFace/data.Paddle
- Benchmarks/Tasks
  - GLUE, SQuAD, sentiment,…
- Hyper-parameter Tuning

Cola subset of Glue, with pretrained models and data from HuggingFace

$gft/examples/eval_examples/model.HuggingFace/language/data.HuggingFace/their_models/glue/cola.sh

Input

```
for model in textattack/distilbert-base-uncased-CoLA textattack/bert-base-uncased-CoLA textattack/distilbert-base-cased-CoLA textattack/roberta-base-CoLA cointegrated/roberta-large-cola-krishna2020 textattack/albert-base-v2-CoLA gchhablani/bert-base-cased-finetuned-cola yoshitomo-matsubara/bert-base-uncased-cola howey/roberta-large-cola textattack/xlnet-base-cased-CoLA isakbos/Q8BERT_COLA_L_512 EhsanAghazadeh/bert-large-uncased-CoLA_A EhsanAghazadeh/bert-large-uncased-CoLA_B 09panesara/distilbert-base-uncased-finetuned-cola yoshitomo-matsubara/bert-large-uncased-cola  textattack/xlnet-large-cased-CoLA Alireza1044/albert-base-v2-cola pmthangk09/bert-base-uncased-glue-cola kamivao/autonlp-cola_gram-208681 mrm8488/deberta-v3-small-finetuned-cola 123abhiALFLKFO/distilbert-base-uncased-finetuned-cola Ahren09/distilbert-base-uncased-finetuned-cola

do

gft_eval --data H:glue,cola  --split val  \

  --eqn 'classify: label ~ sentence'  --model H:$model \

 --metric H:glue,cola  --figure_of_merit matthews_correlation

done
```

Popular models containing "cola" (not SOTA)

Output (Sorted by score; 4 errors removed)

Model	Score	Seconds
howey/roberta-large-cola	0.65	3.93
textattack/roberta-base-CoLA	0.64	1.6
yoshitomo-matsubara/bert-large-uncased-cola	0.63	5.12
mrm8488/deberta-v3-small-finetuned-cola	0.63	1.62
EhsanAghazadeh/bert-large-uncased-CoLA_A	0.61	4.05
yoshitomo-matsubara/bert-base-uncased-cola	0.61	1.58
EhsanAghazadeh/bert-large-uncased-CoLA_B	0.60	4.13
gchhablani/bert-base-cased-finetuned-cola	0.60	1.57
pmthangk09/bert-base-uncased-glue-cola	0.58	1.53
textattack/albert-base-v2-CoLA	0.58	1.59
textattack/distilbert-base-uncased-CoLA	0.57	1.04
kamivao/autonlp-cola_gram-208681	0.56	1.52
Alireza1044/albert-base-v2-cola	0.55	1.71
09panesara/distilbert-base-uncased-finetuned-cola	0.54	0.87
textattack/bert-base-uncased-CoLA	0.53	2.35
123abhiALFLKFO/distilbert-base-uncased-finetuned-cola	0.53	0.9
textattack/xlnet-base-cased-CoLA	0.50	2.23
textattack/distilbert-base-cased-CoLA	0.46	0.88
isakbos/Q8BERT_COLA_L_512	0.00	1.77
textattack/xlnet-large-cased-CoLA	0.00	5.08
cointegrated/roberta-large-cola-krishna2020	-0.65	4.31

What is ``Good'' Stuff? Accuracy? Downloads?

Hyper-Parameter Tuning → Small Gains

Best of Hubs

Model	VAcc	D
C:RoBERTa large, tuned by Yuchen Bian	0.924	
H:textattack/roberta-base-MRPC	0.912	1623
H:textattack/albert-base-v2-MRPC	0.897	175
H:mrm8488/deberta-v3-small-finetuned-mrpc	0.892	30
H:textattack/bert-base-uncased-MRPC	0.877	10,133
H:textattack/distilbert-base-uncased-MRPC	0.858	108
H:ajrae/bert-base-uncased-finetuned-mrpc	0.858	115
C:*gft_fit* example (BERT with no tuning)	0.853	
H:textattack/distilbert-base-cased-MRPC	0.784	122

Table 4: *gft* achieves VAcc (accuracy on validation split) close to distilbert (compressed) models. HuggingFace models were selected using *gft_summary* to find popular models by downloads (D).

SOTA

Source	Test Accuracy
GLUE Leaderboard (L)	0.945
Papers with code (PWC)	0.937
Human Baseline (HB)	0.863

Table 5: SOTA (state-of-the-art) for MRPC (GLUE).

Should not compare accuracy across splits

Best of Hubs are good (but probably not SOTA)

Agenda: Part A

- ✓Strategy
- ✓*gft* Cheatsheet
  - ✓4-5 Functions + 4 Arguments
  - ✓Amazing how much can be done with so little
- ✓Seven Simple Examples

- ➢**More Tasks: MT, ASR, Vision**
- Two Questions
  - How do we *find* the good stuff?
  - How do we *use* the good stuff?
- Why de-emphasize pre-training?
- Conclusions

```
# text-classification: sentiment analysis
echo 'I love you.' | gft_predict --task text-classification
# I love you.   POSITIVE    0.9998705387115479

# text-classification: emotion classification
model=H:AdapterHub/bert-base-uncased-pf-emotion
echo 'I love you.' | gft_predict --model $model --task text-classification
# I love you.   love    0.6005669236183167

# token-classification: NER (Named Entity Recognition)
echo 'I love New York.' | gft_predict --task token-classification
# I love New York.  New/I-LOC:0.9989    York/I-LOC:0.9974
```

More tasks (1 of 4)

✓Text Classification

➢**Token Classification**

- Fill Mask
- Text Generation
- Machine Translation

```
# text-classification: sentiment analysis
echo 'I love you.' | gft_predict --task text-classification
# I love you.   POSITIVE    0.9998705387115479

# text-classification: emotion classification
model=H:AdapterHub/bert-base-uncased-pf-emotion
echo 'I love you.' | gft_predict --model $model --task text-classification
# I love you.   love    0.6005669236183167

# token-classification: NER (Named Entity Recognition)
echo 'I love New York.' | gft_predict --task token-classification
# I love New York.  New/I-LOC:0.9989    York/I-LOC:0.9974

# fill-mask: guess the masked word
echo 'I <mask> you.' | gft_predict --task fill-mask
# I <mask> you.  salute|0.241    miss|0.177  love|0.147  thank|0.060     applaud|0.047
```

More tasks (2 of 4)

- ✓Text Classification
- ✓Token Classification
- ➢**Fill Mask**
- Text Generation
- Machine Translation

More tasks (3 of 4)

```
# text-classification: sentiment analysis
echo 'I love you.' | gft_predict --task text-classification
# I love you.   POSITIVE    0.9998705387115479

# text-classification: emotion classification
model=H:AdapterHub/bert-base-uncased-pf-emotion
echo 'I love you.' | gft_predict --model $model --task text-classification
# I love you.   love    0.6005669236183167

# token-classification: NER (Named Entity Recognition)
echo 'I love New York.' | gft_predict --task token-classification
# I love New York.  New/I-LOC:0.9989    York/I-LOC:0.9974

# fill-mask: guess the masked word
echo 'I <mask> you.' | gft_predict --task fill-mask
# I <mask> you.  salute|0.241    miss|0.177  love|0.147  thank|0.060     applaud|0.047
```

```
# text-generation
echo 'I love ' | gft_predict --task text-generation
# I love you and I will never be forgotten and thank you." I was also
# inspired by all of the students who walked onto campus wearing these
#  teddy I love the idea that you can be anything people ask for you
```

- ✓Text Classification
- ✓Token Classification
- ✓Fill Mask
- ➢**Text Generation**
- Machine Translation

Output is non-deterministic

More tasks (4 of 4)

```
# text-classification: sentiment analysis
echo 'I love you.' | gft_predict --task text-classification
# I love you.   POSITIVE    0.9998705387115479

# text-classification: emotion classification
model=H:AdapterHub/bert-base-uncased-pf-emotion
echo 'I love you.' | gft_predict --model $model --task text-classification
# I love you.   love    0.6005669236183167

# token-classification: NER (Named Entity Recognition)
echo 'I love New York.' | gft_predict --task token-classification
# I love New York.  New/I-LOC:0.9989    York/I-LOC:0.9974

# fill-mask: guess the masked word
echo 'I <mask> you.' | gft_predict --task fill-mask
# I <mask> you.  salute|0.241    miss|0.177  love|0.147  thank|0.060     applaud|0.047

# text-generation
echo 'I love ' | gft_predict --task text-generation
# I love you and I will never be forgotten and thank you." I was also
# inspired by all of the students who walked onto campus wearing these
#  teddy I love the idea that you can be anything people ask for you

# translation
echo 'I love you.' | gft_predict --task translation --model H:Helsinki-NLP/opus-mt-en-fr
# I love you.   Je t'aime.
echo 'I love you.' | gft_predict --task translation --model H:Helsinki-NLP/opus-mt-en-zh
I love you. 我爱你
```

- ✓Text Classification
- ✓Token Classification
- ✓Fill Mask
- ✓Text Generation
- ➢**Machine Translation**

More tasks (4 of 4)

```
# text-classification: sentiment analysis
echo 'I love you.' | gft_predict --task text-classification
# I love you.   POSITIVE    0.9998705387115479

# text-classification: emotion classification
model=H:AdapterHub/bert-base-uncased-pf-emotion
echo 'I love you.' | gft_predict --model $model --task text-classification
# I love you.   love    0.6005669236183167

# token-classification: NER (Named Entity Recognition)
echo 'I love New York.' | gft_predict --task token-classification
# I love New York.  New/I-LOC:0.9989    York/I-LOC:0.9974

# fill-mask: guess the masked word
echo 'I <mask> you.' | gft_predict --task fill-mask
# I <mask> you.  salute|0.241    miss|0.177  love|0.147  thank|0.060     applaud|0.047

# text-generation
echo 'I love ' | gft_predict --task text-generation
# I love you and I will never be forgotten and thank you." I was also
# inspired by all of the students who walked onto campus wearing these
#  teddy I love the idea that you can be anything people ask for you

# translation
echo 'I love you.' | gft_predict --task translation --model H:Helsinki-NLP/opus-mt-en-fr
# I love you.   Je t'aime.
echo 'I love you.' | gft_predict --task translation --model H:Helsinki-NLP/opus-mt-en-zh
I love you. 我爱你
```

- ✓Text Classification
- ✓Token Classification
- ✓Fill Mask
- ✓Text Generation
- ➢**Machine Translation**
  - ➢**English → French**
  - ➢**English → Chinese**

A Model for: English → French

A Model for: English → Chinese

gft_predict –task image-classification

Funny Cat	Cat Chonk
	

Default Model → 1000 Labels

```
url="https://images.all-free-download.com/images/graphicwebp/funny_cat_194619.webp"
echo $url | gft_predict --task H:image-classification
https://images.all-free-download.com/images/graphicwebp/funny_cat_194619.webp   Egyptian cat|0.736  tiger cat|0.039 tabby,
```

The results are more reasonable if we replace the default model with a more appropriate model:

A Model with Cat/Dog Labels

```
url="https://images.all-free-download.com/images/graphicwebp/funny_cat_194619.webp"
echo $url | gft_predict --task H:image-classification --model H:nateraw/vit-base-cats-vs-dogs
https://images.all-free-download.com/images/graphicwebp/funny_cat_194619.webp   cat|1.000   dog|0.000
```

Asirra: a **CAPTCHA** that **exploits interest-aligned manual image categorization**.

J Elson, JR Douceur, J Howell, J Saul - CCS, 2007 - dl.acm.org

History behind Cat/Dog Task

… **Asirra** (Figure 1), a **CAPTCHA** that asks users to identify cats out of a set of 12 photographs of both cats and dogs. **Asirra** is easy … **Asirra**'s **image** database is provided by a novel, mutually …

☆ Save Cite Cited by 581 Related articles All 8 versions

Figure 1: An Asirra challenge. The user selects each of the 12 images that depict cats. As the mouse is hovered over each thumbnail, a larger image and "Adopt me" link appear. "Adopt me" first invalidates the challenge, then takes the user to that animal's page on Petfinder.com.

Figure 6: The differences between cats and dogs are immediately obvious to humans. In many cases, species look similar, with only subtle cues to distinguish them. This makes it a hard vision problem.

Inputs can also come from files (as well as URLs)
There are a few images (and audio files) under $gft/doc/objects

PetImages/0.jpg	PetImages/10000.jpg	PetImages/10001.jpg
		
beans/healthy_test.20.jpg	beans/healthy_test.21.jpg	beans/healthy_test.22.jpg
		

```
model=H:nateraw/vit-base-cats-vs-dogs
ls $gft/doc/objects/images/PetImages/* | sed 3q |
   gft_predict --task H:image-classification --model $model 2>/dev/null
# /mnt/home/kwc/gft7/gft/doc/objects/images/PetImages/0.jpg cat|0.999   dog|0.001
# /mnt/home/kwc/gft7/gft/doc/objects/images/PetImages/10000.jpg cat|1.000   dog|0.000
# /mnt/home/kwc/gft7/gft/doc/objects/images/PetImages/10001.jpg cat|1.000   dog|0.000

model=H:nickmuchi/vit-base-beans
ls $gft/doc/objects/images/beans/* | sed 3q |
   gft_predict --task H:image-classification --model $model 2>/dev/null
# images/beans/healthy_test.20.jpg  healthy|0.996   angular_leaf_spot|0.002 bean_rust|0.002
# images/beans/healthy_test.21.jpg  healthy|0.996   angular_leaf_spot|0.002 bean_rust|0.002
# images/beans/healthy_test.22.jpg  healthy|0.996   angular_leaf_spot|0.002 bean_rust|0.002
```

Task: Speech Recognition (ASR, ctc)

Input from stdin

```
cd $gft/doc/objects/wav/TIMIT;
ls *.WAV | sed 3q | gft_predict --task ASR 2>/dev/null
```

filename	prediction (yhat)
SA1.WAV	SHE HAD YOUR DARK SUIT AND GREASY WASHWATER ALL YEAR
SA2.WAV	DON'T ASK ME TO CARRY AN OILY RAG LIKE THAT
SI1129.WAV	THIS GROUP IS SECULARIST AND THEIR PROGRAMM TENDS TO BE TECHNOLOGICAL

Input from dataset

```
gft_predict --eqn 'ASR:text~file' \
        --data H:timit_asr \
        --split test 2>/dev/null |
    awk -F/ '{print $NF}' | sed 3q
```

filename	gold	prediction (yhat)
SX139.WAV	The bungalow was pleasantly situated near the shore.	THE BUNGALOW WAS PLEASANTLY SITUATED NEAR THE SHORE
SA2.WAV	Don't ask me to carry an oily rag like that.	DON'T ASK ME TO CARRY AN OILY RAG LIKE THAT
SX229.WAV	Are you looking for employment?	ARE YOU LOOKING FOR EMPLOYMENT

--task argument → HuggingFace Pipelines

https://huggingface.co/docs/transformers/v4.16.2/en/main_classes/pipelines

```
# text-classification: sentiment analysis
echo 'I love you.' | gft_predict --task text-classification
# I love you.   POSITIVE    0.9998705387115479

# text-classification: emotion classification
model=H:AdapterHub/bert-base-uncased-pf-emotion
echo 'I love you.' | gft_predict --model $model --task text-classification
# I love you.   love    0.6005669236183167

# token-classification: NER (Named Entity Recognition)
echo 'I love New York.' | gft_predict --task token-classification
# I love New York.  New/I-LOC:0.9989    York/I-LOC:0.9974

# fill-mask: guess the masked word
echo 'I <mask> you.' | gft_predict --task fill-mask
# I <mask> you.  salute|0.241    miss|0.177  love|0.147  thank|0.060     applaud|0.047

# text-generation
echo 'I love ' | gft_predict --task text-generation
# I love you and I will never be forgotten and thank you." I was also
# inspired by all of the students who walked onto campus wearing these
#  teddy I love the idea that you can be anything people ask for you

# translation
echo 'I love you.' | gft_predict --task translation --model H:Helsinki-NLP/opus-mt-en-fr
# I love you.   Je t'aime.
echo 'I love you.' | gft_predict --task translation --model H:Helsinki-NLP/opus-mt-en-zh
I love you. 我爱你
```

task (`str`) — The task defining which pipeline will be returned. Currently accepted tasks are:

- `"audio-classification"`: will return a AudioClassificationPipeline.
- `"automatic-speech-recognition"`: will return a AutomaticSpeechRecognitionPipeline.
- `"conversational"`: will return a ConversationalPipeline.
- `"feature-extraction"`: will return a FeatureExtractionPipeline.
- `"fill-mask"`: will return a FillMaskPipeline:.
- `"image-classification"`: will return a ImageClassificationPipeline.
- `"question-answering"`: will return a QuestionAnsweringPipeline.
- `"table-question-answering"`: will return a TableQuestionAnsweringPipeline.
- `"text2text-generation"`: will return a Text2TextGenerationPipeline.
- `"text-classification"` (alias `"sentiment-analysis"` available): will return a TextClassificationPipeline.
- `"text-generation"`: will return a TextGenerationPipeline:.
- `"token-classification"` (alias `"ner"` available): will return a TokenClassificationPipeline.
- `"translation"`: will return a TranslationPipeline.
- `"translation_xx_to_yy"`: will return a TranslationPipeline.
- `"summarization"`: will return a SummarizationPipeline.
- `"zero-shot-classification"`: will return a ZeroShotClassificationPipeline.

Agenda: Part A

- ✓Strategy
- ✓*gft* Cheatsheet
  - ✓4-5 Functions + 4 Arguments
  - ✓Amazing how much can be done with so little
- ✓Seven Simple Examples
- ✓More Tasks: MT, ASR, Vision
- ➢**Two Questions**
  - How do we *find* the good stuff?
  - How do we *use* the good stuff?
- Why de-emphasize pre-training?
- Conclusions

Two Questions

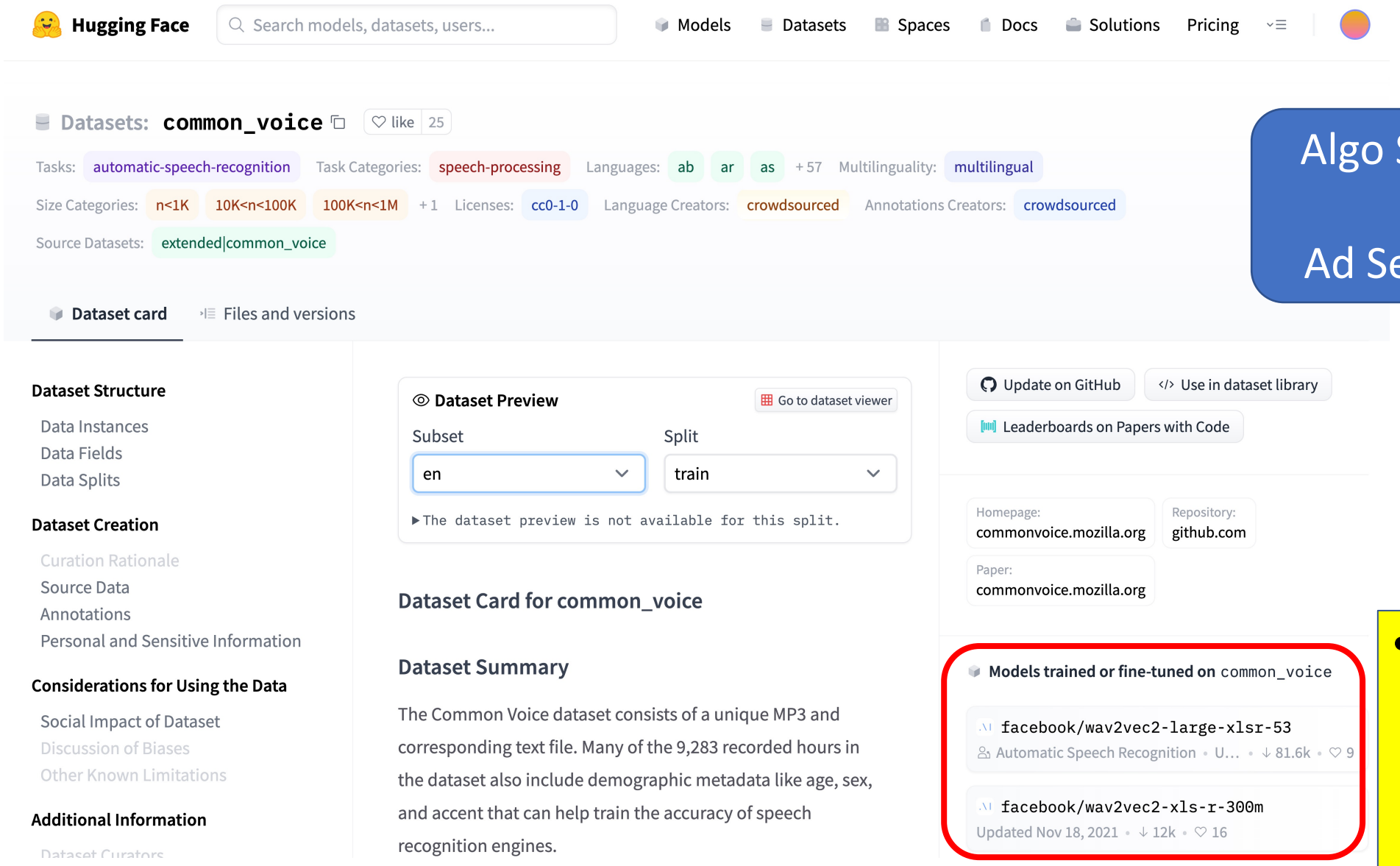
- **How do I find the *good* stuff?**
- And how do I use the *good* stuff?

- Presupposition:
  - Hubs encourage sharing of
    - Datasets
    - Models
    - Tasks

Embarrassment of Riches

- Hubs have 40k models & 4k datasets (increasing 3x per year)
  - Q: How can I find the good stuff?
  - ➢ **A: *gft_summary***

Metrics of Success: How do we find good stuff?



Algo Search: Left Rail
vs
Ad Search: Right Rail

A kind of search engine

- Score dataset by:
  - # of downloads
  - # of likes
  - # of models
  - # of citations

How do I find the good stuff? Popular Datasets? gft_summary –data H:__contains__wiki --topn 20

Dataset	Downloads	Likes	PWC
wikitext	119,031	7	https://paperswithcode.com/dataset/wikitext-2
wikiann	20,021	6	https://paperswithcode.com/dataset/wikiann-1
wiki_snippets	16,550	0	
wikipedia	14,729	8	
wiki_bio	10,135	0	https://paperswithcode.com/dataset/wikibio
wiki_qa	9,670	1	https://paperswithcode.com/dataset/wikiqa
wiki_hop	8,961	0	https://paperswithcode.com/dataset/wikihop
wiki_dpr	5,310	0	
wiki_split	5,260	0	https://paperswithcode.com/dataset/wikisplit
wiki40b	2,618	3	https://paperswithcode.com/dataset/wiki-40b
wikisql	2,135	2	https://paperswithcode.com/dataset/wikisql
wikihow	1,601	0	https://paperswithcode.com/dataset/wikihow
wiki_lingua	1,508	1	https://paperswithcode.com/dataset/wikilingua
kilt_wikipedia	1,421	2	
Tevatron/wikipedia-nq-corpus	1,368	0	
wiki_auto	1,258	0	
wiki_asp	1,192	1	https://paperswithcode.com/dataset/wikiasp
wikicorpus	1,098	0	
wiki_atomic_edits	1,058	3	https://paperswithcode.com/dataset/wikiatomicedits
iapp_wiki_qa_squad	1,033	0	

How do I find the good stuff? Popular Datasets? gft_summary –data H:__contains__ --topn 20

Dataset	Downloads	Likes	PWC
glue	926,082	26	https://paperswithcode.com/dataset/glue
super_glue	306,473	9	https://paperswithcode.com/dataset/superglue
wikitext	119,031	7	https://paperswithcode.com/dataset/wikitext-2
imdb	105,321	8	https://paperswithcode.com/dataset/imdb-movie-reviews
squad	99,954	22	https://paperswithcode.com/dataset/squad
squad_es	72,163	0	https://paperswithcode.com/dataset/squad-es
paws	65,433	1	https://paperswithcode.com/dataset/paws
librispeech_asr	65,325	12	https://paperswithcode.com/dataset/librispeech-1
wmt16	48,479	2	https://paperswithcode.com/dataset/wmt-2016
xnli	41,987	5	https://paperswithcode.com/dataset/xnli
snli	34,045	5	https://paperswithcode.com/dataset/snli
ag_news	30,456	11	https://paperswithcode.com/dataset/ag-news
anli	30,245	3	https://paperswithcode.com/dataset/anli
amazon_polarity	28,584	6	
squad_v2	28,000	3	https://paperswithcode.com/dataset/squad
conll2003	26,012	11	https://paperswithcode.com/dataset/conll-2003
red_caps	25,940	2	https://paperswithcode.com/dataset/redcaps
common_voice	25,033	26	https://paperswithcode.com/dataset/common-voice
stsb_multi_mt	23,866	4	
trec	23,313	3	https://paperswithcode.com/dataset/trecqa

gft_summary: How do I find the good stuff?
Find models associated with a task (ASR)

task → list of models (by downloads)

```
# How do I find the good stuff?
# Return a list of models for a particular task (sorted by downloads)
gft_summary --task H:ASR --model H:__infer__ 2>/dev/null | head
# task: AutomaticSpeechRecognition --> 1643 models
# task: AutomaticSpeechRecognition  model: facebook/wav2vec2-base-960h  downloads: 1092953  like
# task: AutomaticSpeechRecognition  model: facebook/hubert-large-ls960-ft   downloads: 65234
# task: AutomaticSpeechRecognition  model: facebook/wav2vec2-large-960h-lv60-self   downloads: 6
# task: AutomaticSpeechRecognition  model: facebook/wav2vec2-large-xlsr-53  downloads: 55869
# task: AutomaticSpeechRecognition  model: hf-internal-testing/processor_with_lm    downloads: 2
# task: AutomaticSpeechRecognition  model: pyannote/voice-activity-detection    downloads: 1895
# task: AutomaticSpeechRecognition  model: patrickvonplaten/wavlm-libri-clean-100h-base-plus
# task: AutomaticSpeechRecognition  model: facebook/wav2vec2-large-960h downloads: 13622    like
# task: AutomaticSpeechRecognition  model: facebook/s2t-small-librispeech-asr   downloads: 11808
```

gft_summary: How do I find the good stuff?
Find models associated with a task (ASR)

task → list of models (by downloads)

```
# How do I find the good stuff?
# Return a list of models for a particular task (sorted by downloads)
gft_summary --task H:ASR --model H:__infer__ 2>/dev/null | head
# task: AutomaticSpeechRecognition --> 1643 models
# task: AutomaticSpeechRecognition  model: facebook/wav2vec2-base-960h  downloads: 1092953  like
# task: AutomaticSpeechRecognition  model: facebook/hubert-large-ls960-ft  downloads: 65234
# task: AutomaticSpeechRecognition  model: facebook/wav2vec2-large-960h-lv60-self  downloads:
# task: AutomaticSpeechRecognition  model: facebook/wav2vec2-large-
# task: AutomaticSpeechRecognition  model: hf-internal-testing/proc
# task: AutomaticSpeechRecognition  model: pyannote/voice-activity-
# task: AutomaticSpeechRecognition  model: patrickvonplaten/wavlm-l
# task: AutomaticSpeechRecognition  model: facebook/wav2vec2-large-
# task: AutomaticSpeechRecognition  model: facebook/s2t-small-libri
```

dataset → list of models (by downloads)

```
# How do I find the good stuff?
# Return a list of models for a particular dataset (sorted by downloads)
gft_summary --data H:common_voice,en --model H:__infer__ 2>/dev/null | head
# dataset: common_voice,en  splits: train: 564337 rows, test: 16164 rows, other: 169895 rows, i
# dataset: common_voice,en  split: train  columns: client_id, path, audio, sentence, up_votes,
# dataset: common_voice --> 542 models
# dataset: common_voice model: facebook/wav2vec2-large-xlsr-53  downloads: 55869  likes: 12
# dataset: common_voice model: jonatasgrosman/wav2vec2-large-xlsr-53-english  downloads: 11633
# dataset: common_voice model: facebook/wav2vec2-xls-r-300m downloads: 10156  likes: 16
# dataset: common_voice model: vumichien/wav2vec2-large-xlsr-japanese-hiragana  downloads: 8454
# dataset: common_voice model: jonatasgrosman/wav2vec2-large-xlsr-53-german downloads: 6338 like
# dataset: common_voice model: jonatasgrosman/wav2vec2-large-xlsr-53-russian  downloads: 4689
# dataset: common_voice model: facebook/wav2vec2-large-xlsr-53-german  downloads: 4498 likes:
```

gft_summary: How do I find the good stuff?
Find popular models associated with a dataset

```
# find some models associated with a dataset (and sort by downloads)
gft_summary --data H:emotion --model H:__infer__ 2>/dev/null | head
# dataset: emotion  split: train    columns: text, label
# dataset: emotion  labels: sadness, joy, love, anger, fear, surprise
# dataset: emotion --> 69 models
# dataset: emotion  model: bhadresh-savani/distilbert-base-uncased-emotion  downloads: 505632
# dataset: emotion  model: nateraw/bert-base-uncased-emotion    downloads: 5836 likes: 1
# dataset: emotion  model: mrm8488/t5-base-finetuned-emotion    downloads: 3927 likes: 3
# dataset: emotion  model: mrm8488/t5-small-finetuned-emotion   downloads: 1580 likes: 0
# dataset: emotion  model: bhadresh-savani/roberta-base-emotion downloads: 1432 likes: 0
# dataset: emotion  model: bhadresh-savani/bert-base-uncased-emotion    downloads: 587  likes: 1
# dataset: emotion  model: lewiswatson/distilbert-base-uncased-finetuned-emotion    downloads: 2
```

gft_summary: How do I find the good stuff? Find popular models associated with a task (MT)

```
# ditto, but for machine translation
gft_summary --task H:translation --model H:__infer__ 2>/dev/null | head
# task: Translation --> 1470 models
# task: Translation model: Helsinki-NLP/opus-mt-zh-en   downloads: 7606505  likes: 17
# task: Translation model: t5-small downloads: 986817   likes: 5
# task: Translation model: t5-base  downloads: 812813   likes: 32
# task: Translation model: Helsinki-NLP/opus-mt-en-de   downloads: 721507   likes: 2
# task: Translation model: Helsinki-NLP/opus-mt-en-ROMANCE  downloads: 435634   likes: 1
# task: Translation model: Helsinki-NLP/opus-mt-ar-en   downloads: 185275   likes: 3
# task: Translation model: Helsinki-NLP/opus-mt-es-en   downloads: 175059   likes: 6
# task: Translation model: Helsinki-NLP/opus-mt-de-en   downloads: 144799   likes: 4
# task: Translation model: Helsinki-NLP/opus-mt-fr-en   downloads: 141266   likes: 1
```

Embarrassment of Riches

- ✓Hubs have 30k models & 3k datasets (increasing 3x per year)
  - ✓Q: How can I find the good stuff?
  - ✓A: *gft_summary*
    - ✓ __contains__
      - ✓ Find popular models/datasets that contain substring
    - ✓ __infer__
      - ✓ Find popular models that are associated with task/dataset
  - ✓Examples:
    - ✓gft_summary --data H:__contains__emotion --topn 5
    - ✓gft_summary --model H:__contains__emotion --topn 5
    - ✓gft_summary --task H:classify --model H:__infer__ --topn 5
    - ✓gft_summary --data H:emotion --model H:__infer__ --topn 5

What counts as ``good'' stuff?

- Academic Metrics:
  - Peer Review (Citations)
- Social Media Metrics:
  - Popular (Downloads)
  - Tweets / Blogs
- SOTA: State of the art
  - Papers with Code
  - Leaderboards
- Search Engine Optimization (SEO)
  - Adversarial Game (Arbitrage):
    - Better Web Search → ``Better'' SEOs → Better Web Search

Large Model (RoBERTa Large)

Many Downloads, but not SOTA

Compressed (distilbert)

Ease-of-Use

Model	VAcc	Dload
C:Best of Yuchen Bian's models	0.924	
H:textattack/roberta-base-MRPC	0.912	1623
H:textattack/albert-base-v2-MRPC	0.897	175
H:mrm8488/deberta-v3-small-finetuned-mrpc	0.892	30
H:textattack/bert-base-uncased-MRPC	0.877	10,133
H:textattack/distilbert-base-uncased-MRPC	0.858	108
H:ajrae/bert-base-uncased-finetuned-mrpc	0.858	115
C:*gft_fit* example (with default settings)	0.853	
H:textattack/distilbert-base-cased-MRPC	0.784	122

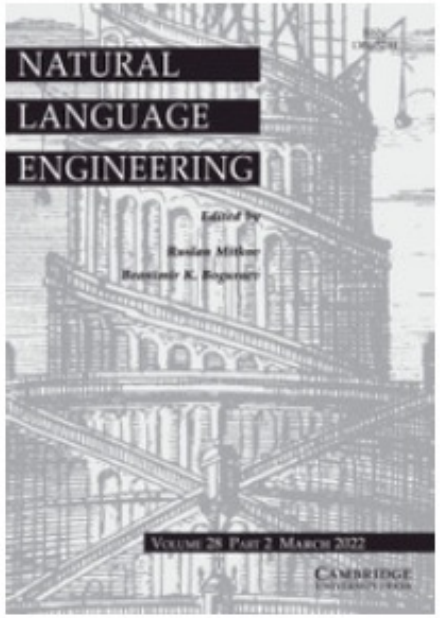
Table 4: *gft* achieves VAcc (accuracy on validation split) close to distilbert (compressed) models. HuggingFace models were selected using *gft_summary* to find popular models by downloads (Dload).

Source	Test Accuracy
GLUE Leaderboard (L)	0.945
Papers with code (PWC)	0.937
Human Baseline (HB)	0.863

Table 5: SOTA (state-of-the-art) for MRPC (GLUE). See footnote 11 for PWC, and 12 for L & HB.

SOTA-Chasing: Leaderboards Considered Harmful

SOTA-chasing refers to papers that report SOTA numbers, but contribute little of lasting value to the literature. The point is the pointlessness.



Natural Language Engineering

Emerging Trends: SOTA-Chasing

Published online by Cambridge University Press: **08 February 2022**

Kenneth Ward Church and Valia Kordoni

Article | Figures | Metrics

Save PDF | Share | Cite | Rights & Permissions

Abstract

PWC: https://paperswithcode.com/dataset/glue

Search

Browse State-of-the-Art | Datasets | Methods | More | We are hiring!

Aα Texts

GLUE (General Language Understanding Evaluation benchmark)

Edit

Introduced by Wang et al. in GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding

General Language Understanding Evaluation (**GLUE**) benchmark is a collection of nine natural language understanding tasks, including single-sentence tasks CoLA and SST-2, similarity and paraphrasing tasks MRPC, STS-B and QQP, and natural language inference tasks MNLI, QNLI, RTE and WNLI.

Source: Align, Mask and Select: A Simple Method for Incorporating Commonsense Knowledge into Language Representation Models

Homepage

GLUE

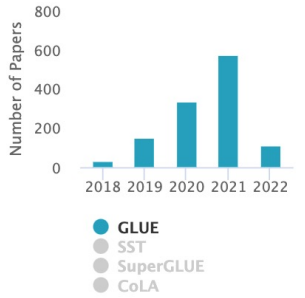
Source: https://gluebenchmark.com/.

Benchmarks

Edit

Trend	Task	Dataset Variant	Best Model	Paper	Code
	Text Classification	GLUE	roberta-base-finetuned-sst2		
	Sentiment Analysis	SST-2 Binary classification	SMART-RoBERTa Large		
	Semantic Textual Similarity	STS Benchmark	SMART-RoBERTa Large		

Usage



PWC ≠ GLUE Leaderboard

	Rank	Name	Model	URL	Score	CoLA	SST-2	MRPC	STS-B	QQP	MNLI-m
	1	JDExplore d-team	Vega v1		91.3	73.8	97.9	94.5/92.6	93.5/93.1	76.7/91.1	92.1
	2	Microsoft Alexander v-team	Turing NLR v5		91.2	72.6	97.6	93.8/91.7	93.7/93.3	76.4/91.1	92.6
	3	DIRL Team	DeBERTa + CLEVER		91.1	74.7	97.6	93.3/91.1	93.4/93.1	76.5/91.0	92.1
	4	ERNIE Team - Baidu	ERNIE		91.1	75.5	97.8	93.9/91.8	93.0/92.6	75.2/90.9	92.3
	5	AliceMind & DIRL	StructBERT + CLEVER		91.0	75.3	97.7	93.9/91.9	93.5/93.1	75.6/90.8	91.7
	6	DeBERTa Team - Microsoft	DeBERTa / TuringNLRv4		90.8	71.5	97.5	94.0/92.0	92.9/92.6	76.2/90.8	91.9
	7	HFL iFLYTEK	MacALBERT + DKM		90.7	74.8	97.0	94.5/92.6	92.8/92.6	74.7/90.6	91.3
+	8	PING-AN Omni-Sinitic	ALBERT + DAAF + NAS		90.6	73.5	97.2	94.0/92.0	93.0/92.4	76.1/91.0	91.6
	9	T5 Team - Google	T5		90.3	71.6	97.5	92.8/90.4	93.1/92.8	75.1/90.6	92.2
	10	Microsoft D365 AI & MSR AI & GATECH	MT-DNN-SMART		89.9	69.5	97.5	93.7/91.6	92.9/92.5	73.9/90.2	91.0
+	11	Huawei Noah's Ark Lab	NEZHA-Large		89.8	71.7	97.3	93.3/91.0	92.4/91.9	75.2/90.7	91.5
+	12	Zihang Dai	Funnel-Transformer (Ensemble B10-10-10H1024)		89.7	70.5	97.5	93.4/91.2	92.6/92.3	75.4/90.7	91.4
+	13	ELECTRA Team	ELECTRA-Large + Standard Tricks		89.4	71.7	97.1	93.1/90.7	92.9/92.5	75.6/90.8	91.3
	14	Shi-Wen Ni	DropSAT-RoBERTa-large		88.8	70.2	96.7	92.6/90.1	92.1/91.8	75.1/90.5	91.1
	15	DropAttack Team	DropAK-ELECTRA-large		88.7	70.4	95.8	92.6/90.1	91.2/91.1	75.1/90.5	91.1
	16	R-AT Paper	RoBERTa-large + R-AT		88.6	69.0	96.7	92.5/90.0	92.1/91.8	75.0/90.5	91.1
+	17	Microsoft D365 AI & UMD	FreeLB-RoBERTa (ensemble)		88.4	68.0	96.8	93.1/90.8	92.3/92.1	74.8/90.3	91.1
	18	Junjie Yang	HIRE-RoBERTa		88.3	68.6	97.1	93.0/90.7	92.4/92.0	74.3/90.2	90.7
+	19	Shiwen Ni	ELECTRA-large-M (bert4keras)		88.3	69.3	95.8	92.2/89.6	91.2/91.1	75.1/90.5	91.1
	20	Facebook AI	RoBERTa		88.1	67.8	96.7	92.3/89.8	92.2/91.9	74.3/90.2	90.8
+	21	Microsoft D365 AI & MSR AI	MT-DNN-ensemble		87.6	68.4	96.5	92.7/90.3	91.1/90.7	73.7/90.0	87.9

How do I find ``good'' tasks?

- ✓``Good'' tasks are like ``good'' stuff
  - ✓Academic Metrics:
    - ✓Peer Review (Citations)
  - ✓Social Media Metrics:
    - ✓Popular (Downloads)
    - ✓Tweets / Blogs
  - ✓SOTA: State of the art
    - ✓Papers with Code
    - ✓Leaderboards
  - ✓Search Engine Optimization (SEO)
    - ✓Adversarial Game (Arbitrage):
      - ✓ Better Web Search → ``Better'' SEOs → Better Web Search

Equation Keywords ≈ Pipeline Tasks

Task	Subtask	Dataset	Equation
GLUE	COLA	H:glue,cola	$classify: label \sim sentence$
SQuAD 1.0		H:squad	$classify\_spans: answers \sim question + context$
SQuAD 2.0		H:squad_v2	$classify\_spans: answers \sim question + context$
CONLL2003	POS	H:conll2003	$classify\_tokens: pos\_tags \sim tokens$
	NER	H:conll2003	$classify\_tokens: ner\_tags \sim tokens$
TIMIT		H:timit_asr	$ctc: text \sim audio$
Amazon Reviews		H:amazon_reviews_multi	$classify: label \sim question + sentence$
VAD		C:$gft/datasets/VAD/VAD	$regress: Valence + Arousal + Dominance \sim Word$

What are the most popular tasks?
30k Models → Pipeline Tasks

(computed from $gft/gft_internals/huggingface_hub/huggingface_datasets.txt)

Models	Pipeline Task
9537	None
3949	text-generation
3143	text-classification (*classify*)
2390	fill-mask
2253	text2text-generation
1508	automatic-speech-recognition (*ctc*)
1460	translation
1340	token-classification (*classify_tokens*)
1042	conversational
983	question-answering (*classify_spans*)
982	feature-extraction
338	sentence-similarity
320	summarization
261	image-classification

f_{pre}: Pre-trained Model

Models	Pipeline Task
160	text-to-speech
68	zero-shot-classification
55	audio-to-audio
54	audio-classification
26	table-question-answering
12	object-detection
11	image-segmentation
9	text-to-image
4	structured-data-classification
4	image-to-text
2	voice-activity-detection
1	zero-shot-image-classification
1	speech-segmentation
1	protein-folding
1	pipeline_tag

30k Models → Pipeline Tasks

(computed from $gft/gft_internals/huggingface_hub/huggingface_datasets.txt)

Models	**Pipeline Task**
9537	None
3949	text-generation
3143	text-classification (*classify*)
2390	fill-mask
2253	text2text-generation
1508	automatic-speech-recognitic
1460	translation
1340	token-classification (*classify_*
1042	conversational
983	question-answering (*classify*
982	feature-extraction
338	sentence-similarity
320	summarization
261	image-classification

# of Models	Task	Models
3949	text-generation	distilgpt2, gpt2, EleutherAI/gpt-neo-1.3B
3143	text-classification	cross-encoder/ms-marco-MiniLM-L-12-v2, distilbert-base-uncased-finetuned-sst-2-eng
2390	fill-mask bert-base-uncased	distilbert-base-uncased, roberta-base
2253	text2text-generation	facebook/m2m100_418M, facebook/mbart-large-50-one-to-many-mmt, google/mt5-base
1508	ctc	facebook/wav2vec2-base-960h, facebook/hubert-large-ls960-ft, facebook/wav2vec2-lar
1460	translation	Helsinki-NLP/opus-mt-zh-en, t5-small, t5-base
1340	classify	xlm-roberta-large-finetuned-conll03-english, classla/bcms-bertic-ner, dslim/bert-base-N
1042	conversational	microsoft/DialoGPT-small, microsoft/DialoGPT-medium, facebook/blenderbot_small-90M
983	classify_spans	deepset/roberta-base-squad2, distilbert-base-cased-distilled-squad, bert-large-uncased
982	feature-extraction	feature-extraction, openai/clip-vit-base-patch32, facebook/bart-base, monsoon-nlp/hind
338	sentence-similarity	sentence-transformers/multi-qa-MiniLM-L6-cos-v1, sentence-transformers/paraphrase-M L6-v2

Two Questions

✓How do I ***find*** the *good* stuff?

➢**And how do I *use* the *good* stuff?**

➢**A: gft_predict**

Question: How do I ***use*** the good stuff?
Answer: *gft_predict*

$f(x) \rightarrow \hat{y}$

No model specified

```
# text-classification: sentiment analysis
echo 'I love you.' | gft_predict --task text-classification
# I love you.   POSITIVE    0.9998705387115479
```

x $\hat{y}$ Score

Input: *I love you*
Output: Depends on Model (among other things)

- Task: classify
- Models:
  - Sentiment
    - Expected output label: positive
  - Fake News:
    - Expected output label: not fake (real)
  - Spam/Ham:
    - Expected output label: not spam (ham)

```
for model in …
 do
   echo I love you |
   gft_predict \
      --task classify \
      --model $model
 done
```

Scores could be more confident

Many of these models produce reasonable results

I love you is **positive**

Predicted Label	Score	Model	Labels for Model
positive	0.512	SetFit/deberta-v3-large__sst2__train-16-7	negative, positive
POSITIVE	0.871	ayameRushia/roberta-base-indonesian-sentiment-analysis-smsa	POSITIVE, NEUTRAL, NEGATIVE
positive	0.807	SetFit/distilbert-base-uncased__sst2__train-32-2	negative, positive
positive	0.999	AdapterHub/bert-base-uncased-pf-sst2	negative, positive
positive	0.917	SetFit/deberta-v3-large__sst2__train-32-1	negative, positive
positive	0.999	moshew/tiny-bert-aug-sst2-distilled	negative, positive
positive	0.651	rohansingh/autonlp-Fake-news-detection-system-29906863	negative, positive
5 stars	0.872	tomato/sentiment_analysis	1 star, 2 stars, 3 stars, 4 stars, 5 stars
5 stars	0.424	cmarkea/distilcamembert-base-sentiment	1 star, 2 stars, 3 stars, 4 stars, 5 stars
5 stars	0.872	nlptown/bert-base-multilingual-uncased-sentiment	1 star, 2 stars, 3 stars, 4 stars, 5 stars

But..

I love you is **fake news**

Predicted Label	Score	Model	Labels for Model
Fake	0.998	yaoyinnan/bert-base-chinese-covid19	Neutral, Fake, Real
Fake	0.986	yaoyinnan/roberta-fakeddit	Fake, Real
fake	0.958	Qiaozhen/fake-news-detector	real, fake
FAKE	0.959	Narrativaai/fake-news-detection-spanish	REAL, FAKE

I love you is both **spam** and **ham**

Predicted Label	Score	Model	Labels for Model
spam	0.826	SetFit/distilbert-base-uncased__enron_spam__all-train	ham, spam
not spam	1.000	sureshs/distilbert-large-sms-spam	not spam, spam

Debugging, Confusion Matrices & Error Analysis

In addition to producing a score with *gft_eval*, suppose we want to do some deep dives to look at particular errors. The code in Listing 19 will create a confusion matrix based on the validation split.

```
1 f=H:bhadresh-savani/distilbert-base-uncased-emotion
2 gft_predict --eqn 'classify:label~text' --model $f \
3         --data H:emotion --split val > /tmp/pred
4 cut -f2,3 < /tmp/pred | sort | uniq -c | sort -nr > /tmp/conf
```

Listing 19. Code to create confusion matrix

gft_predict outputs TSV (tab separated values) with 4 columns:

1. Input, x
2. Gold label, y
3. Predicted label, $\hat{y}$
4. Score

The cut statement on line 4 in Listing 19 selects y and $\hat{y}$. The sort and uniq statements count the number of confusions, producing the confusion matrix shown in Table 6. Standard Unix tools such as grep (or AWK) can be used to find more details for particular confusions.

Gold Labels	**Predicted Labels**, $\hat{y}$					
y	sadness	joy	love	anger	fear	surprise
sadness	530	1	1	7	11	0
joy	1	670	26	0	3	4
love	4	21	153	0	0	0
anger	10	4	1	254	6	0
fear	5	2	0	3	194	8
surprise	1	3	0	0	10	67

Table 6. *Confusion matrix from Listing 19*

Inference: *gft_predict*

- gft_predict
  - Input from a dataset or *stdin*
  - Output to *stdout*
- Arguments (many are optional)
  - --data, --split, --model, --eqn, --task
  - --eqn $task$: $lhs \sim rhs$
    - Variables in lhs and rhs
    - refer to columns in dataset
- Since the terminology for tasks is currently in a state of flux,
  - gft supports a number of aliases
  - (shown in parentheses)

- Tasks (aliases in parenthesis)
  1. classify (text-classification)
  2. classify_tokens
     - (token-classification)
  3. classify_spans
     - (QA, question-answering)
  4. classify_audio (audio-classification)
  5. classify_images (image-classification)
  6. regress
  7. text-generation
  8. MT (translation)
  9. ASR
     - (ctc, automatic-speech-recognition)
  10. fill-mask

Agenda: Part A

- ✓Strategy
- ✓*gft* Cheatsheet
  - ✓4-5 Functions + 4 Arguments
  - ✓Amazing how much can be done with so little
- ✓Seven Simple Examples
- ✓More Tasks: MT, ASR, Vision
- ✓Two Questions
  - ✓How do we *find* the good stuff?
  - ✓How do we *use* the good stuff?
- ➢**Why de-emphasize pre-training?**
- Conclusions

Why De-emphasize Pre-Training?

Standard 3-Step Recipe

Step	gft Support	Description	Time	Hardware
1		Pre-Training	Days/Weeks	Large GPU Cluster
2	*gft_fit*	Fine-Tuning	Hours/Days	1+ GPUs
3	*gft_predict*	Inference	Seconds/Minutes	0+ GPUs

One-Time

$$$$ (Cost)

Recurring

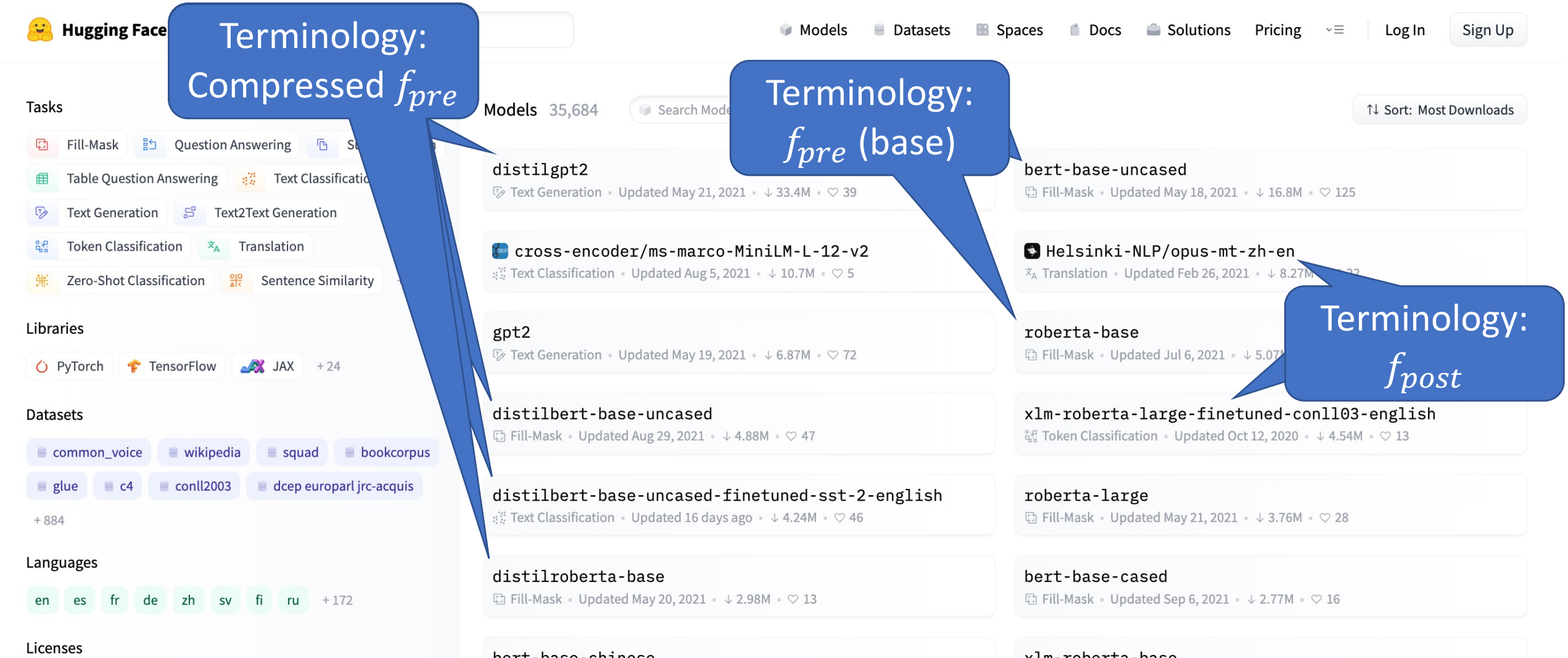
$

less cost; more profit

Models

Terminology: f, f_{pre}, f_{post}

https://huggingface.co/models

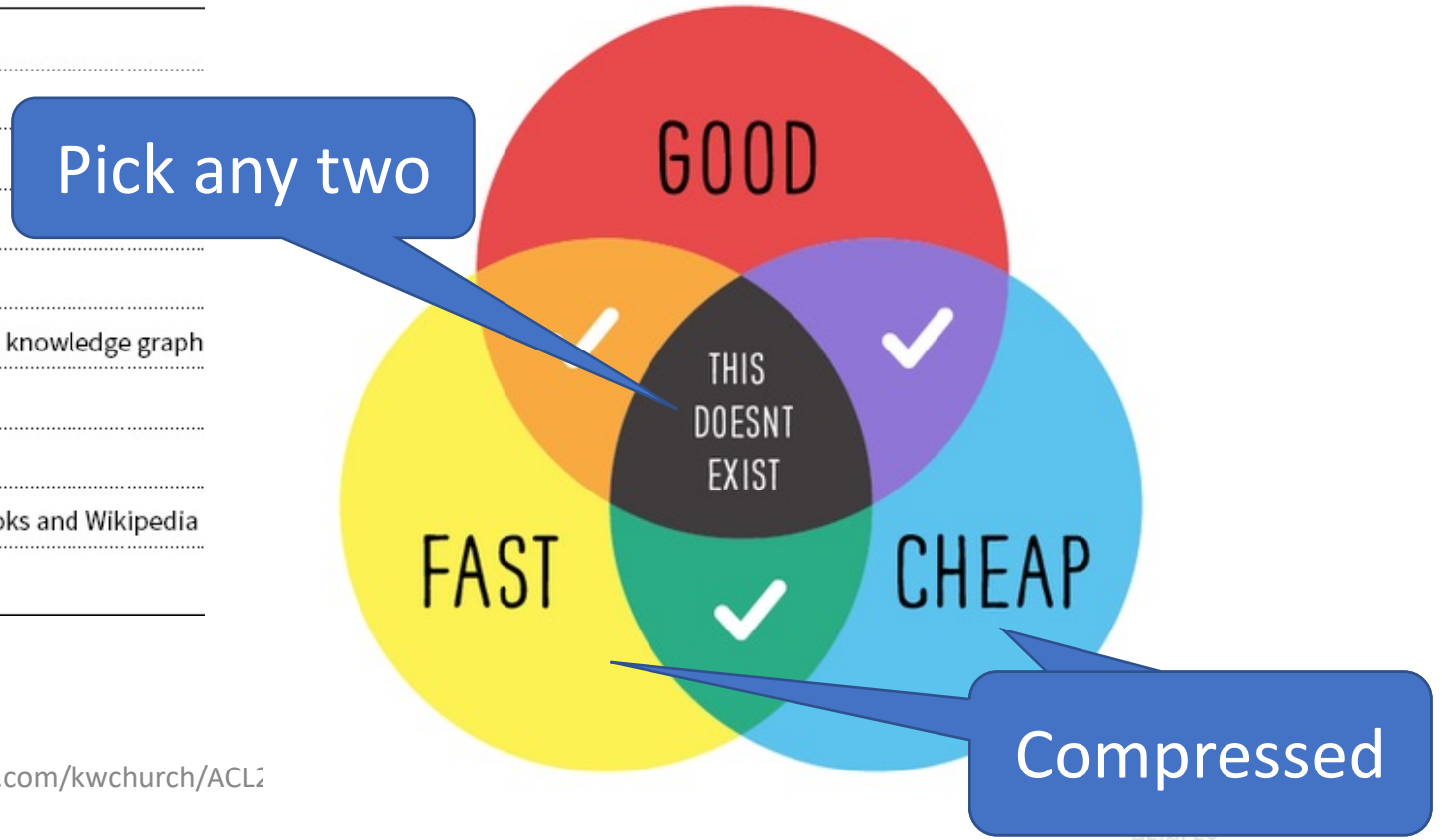


Model Compression (Distillation): Important for Commercial Practice

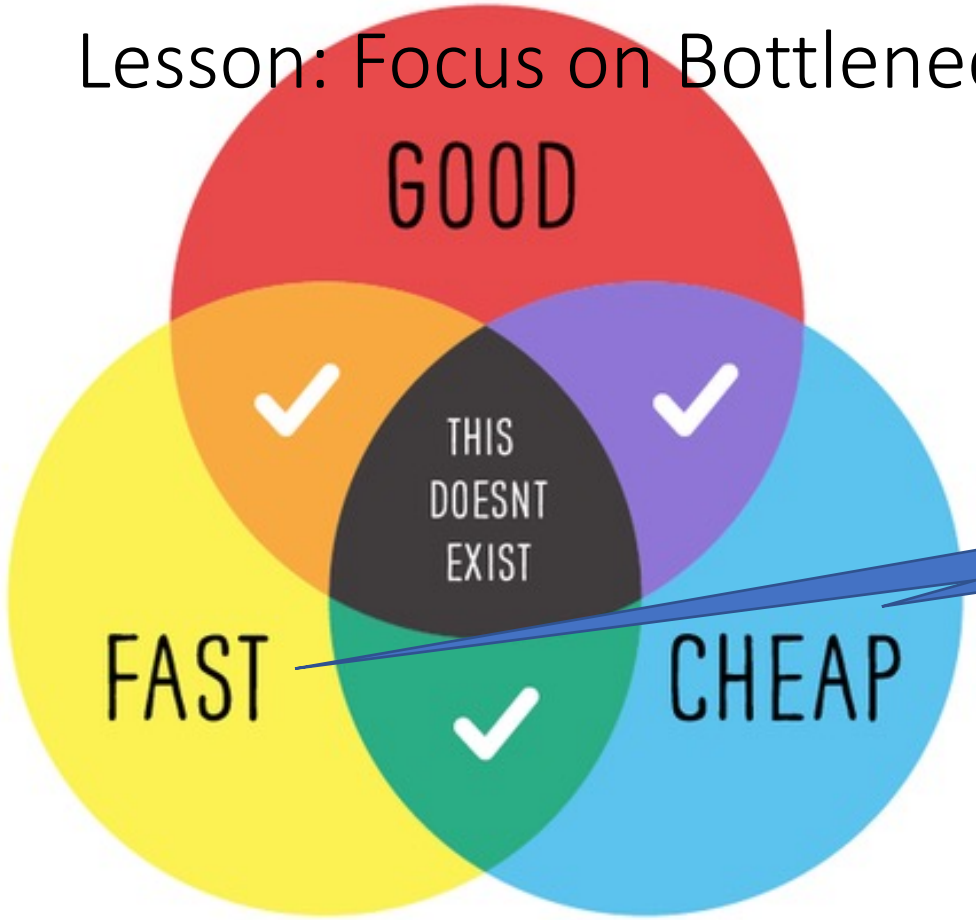
Popular Choices on Hubs

Pre-trained (foundation) model	Parameters	Training data
ResNet-50 (He *et al.* 2016)	23M	14M images from ImageNet
VT (Wu *et al.* 2020)	11.7–21.9M	14M images from ImageNet
Wav2vec (Baevski *et al.* 2020)	95–317M	960 hours from LibriSpeech
BERT (Devlin *et al.* 2019)	110–340M	3.3B words from Books/Wiki
ERNIE 2.0 (Sun *et al.* 2020)	110–340M	7.9B en + 15B zh tokens
ERNIE 3.0 (Sun *et al.* 2021)	10B	375B tokens of text, as well as knowledge graph
RoBERTa (Liu *et al.* 2019)	110M	160GBs of text
GPT-2 (Radford *et al.* 2019)	1.5B	40GBs of text
GPT-3 (Brown *et al.* 2020)	125M–175B	1TB from Common Crawl, Books and Wikipedia
XLM-RoBERTa (Conneau *et al.* 2020)	12–16B	2.5TBs from many languages

Model Compression: Trade-offs: Size vs. Accuracy



Lesson: Focus on Bottlenecks



Bottleneck for Global Warming:
$cost(transportation) \gg 5\ cars$
$cost(smart\ speakers) \gg 5\ cars$

Commercial Practice:
$cost(inference) \gg cost(training)$

- Inference Costs: Recurring costs
  - Inference Costs ≫ 5 cars
    - Deploy a model to 1B people
    - Who use it every day (for years)
- If the application goes viral (success)
  - $variable\ costs \gg fixed\ costs$
  - $cost(inference) \gg cost(training)$

~~Base/Foundation/Pre-Trained Models~~

Recommendation: Download f_{pre} from Hubs

Some Popular Pre-Trained Models: f_{pre}

Pre-trained (foundation) model	Parameters	Training data
ResNet-50 (He *et al.* 2016)	23M	14M images from ImageNet
VT (Wu *et al.* 2020)	11.7–21.9M	14M images from ImageNet
Wav2vec (Baevski *et al.* 2020)	95–317M	960 hours from LibriSpeech
BERT (Devlin *et al.* 2019)	110–340M	3.3B words from Books/Wiki
ERNIE 2.0 (Sun *et al.* 2020)	110–340M	7.9B en + 15B zh tokens
ERNIE 3.0 (Sun *et al.* 2021)	10B	375B tokens of text, as well as knowledge graph
RoBERTa (Liu *et al.* 2019)	110M	160GBs of text
GPT-2 (Radford *et al.* 2019)	1.5B	40GBs of text
GPT-3 (Brown *et al.* 2020)	125M–175B	1TB from Common Crawl, Books and Wikipedia
XLM-RoBERTa (Conneau *et al.* 2020)	12–16B	2.5TBs from many languages

Some Popular Datasets for Pre-Training

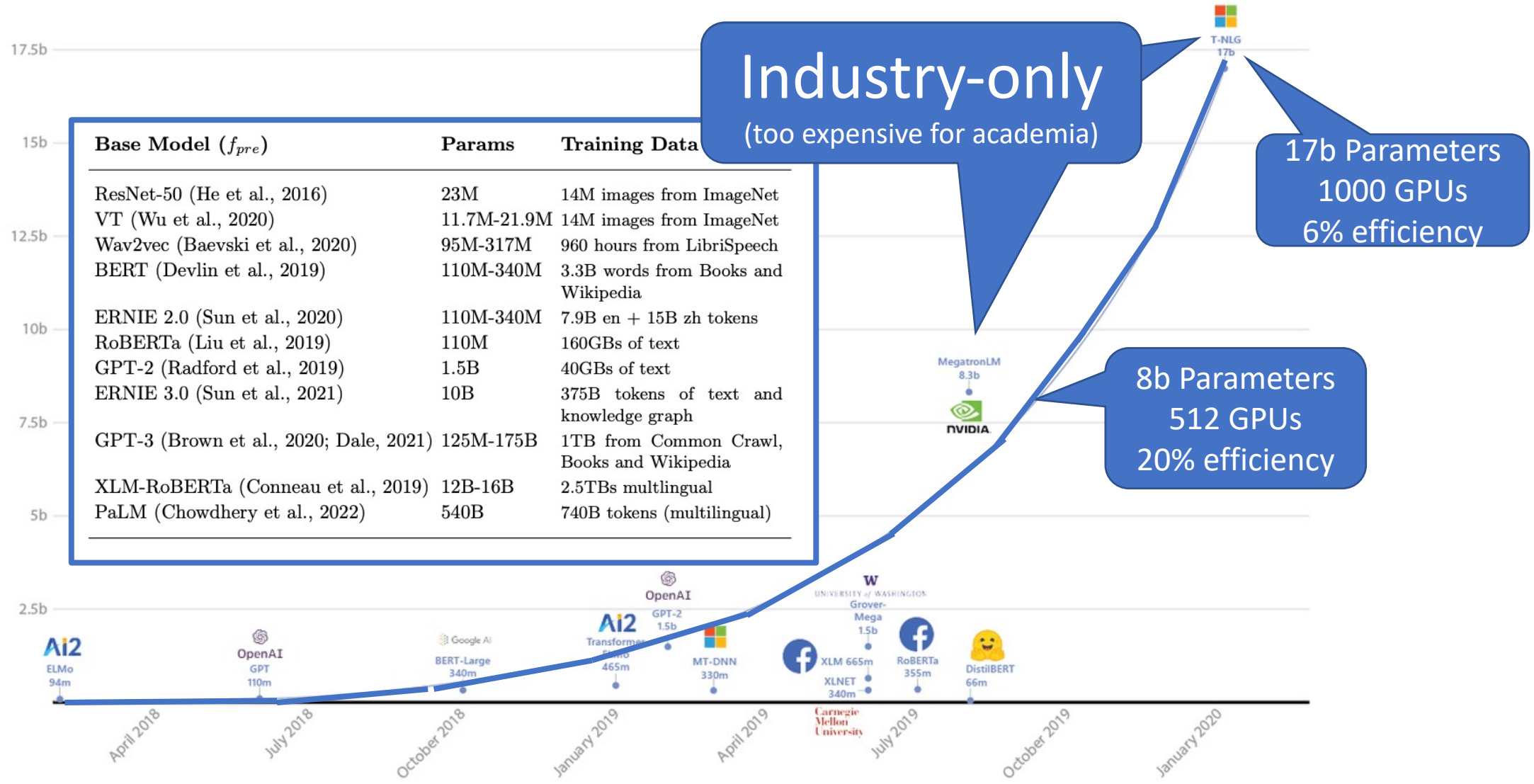
Dataset	Description
Billion Word Benchmark (Chelba *et al.* 2013)	A billion words of English
Common Crawl (Buck, Heafield, and van Ooyen 2014)	https://github.com/commoncrawl
Book Corpus (Zhu *et al.* 2015)	Speech with text
ImageNet (Deng *et al.* 2009)	14M images, annotated with 21k classes
LibriSpeech (Panayotov *et al.* 2015)	960 hours of speech with text
LJ Speech	https://keithito.com/LJ-Speech-Dataset/
AISHELL-2 (Du *et al.* 2018)	https://protect-eu.mimecast.com/s/_t4dC0Vqq sjz5YGswPwKH?domain=aishelltech.com

DDI → Don't Do It (yourself)

Bigger Models are Better (but less efficient)

Data Source: Microsoft blog (https://syncedreview.com/2020/02/12/17-billion-parameters-microsoft-deepspeed-breeds-worlds-largest-nlp-model/) and Kunle Olukton's presentation at ScaledML

Base Model (f_{pre})	Params	Training Data
ResNet-50 (He et al., 2016)	23M	14M images from ImageNet
VT (Wu et al., 2020)	11.7M-21.9M	14M images from ImageNet
Wav2vec (Baevski et al., 2020)	95M-317M	960 hours from LibriSpeech
BERT (Devlin et al., 2019)	110M-340M	3.3B words from Books and Wikipedia
ERNIE 2.0 (Sun et al., 2020)	110M-340M	7.9B en + 15B zh tokens
RoBERTa (Liu et al., 2019)	110M	160GBs of text
GPT-2 (Radford et al., 2019)	1.5B	40GBs of text
ERNIE 3.0 (Sun et al., 2021)	10B	375B tokens of text and knowledge graph
GPT-3 (Brown et al., 2020; Dale, 2021)	125M-175B	1TB from Common Crawl, Books and Wikipedia
XLM-RoBERTa (Conneau et al., 2019)	12B-16B	2.5TBs multlingual
PaLM (Chowdhery et al., 2022)	540B	740B tokens (multilingual)

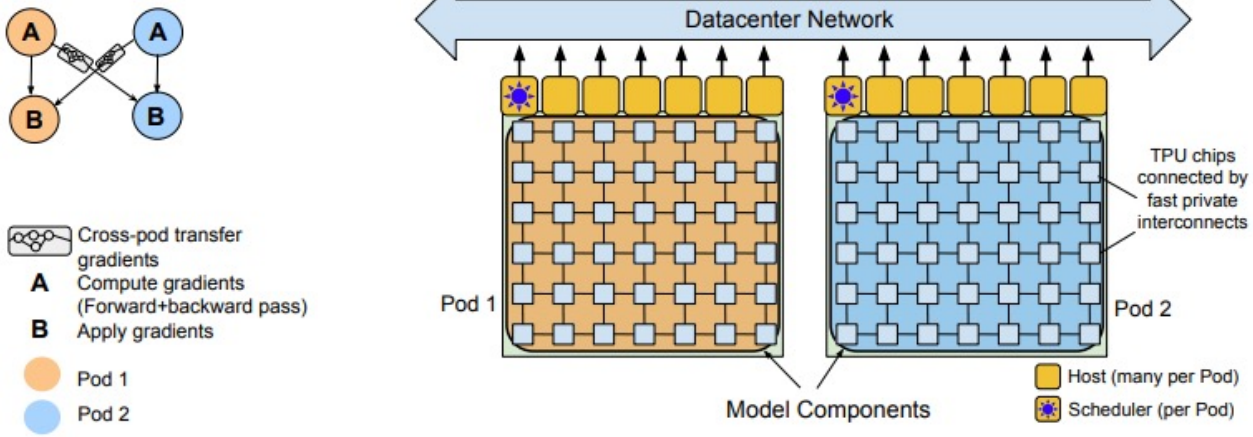


540B Parameters:

https://arxiv.org/pdf/2204.02311.pdf

Industry-only
(too expective for academia)

Model	# of Parameters (in billions)	Accelerator chips	Model FLOPS utilization
GPT-3	175B	V100	21.3%
Gopher	280B	4096 TPU v3	32.5%
Megatron-Turing NLG	530B	2240 A100	30.2%
PaLM	540B	6144 TPU v4	46.2%



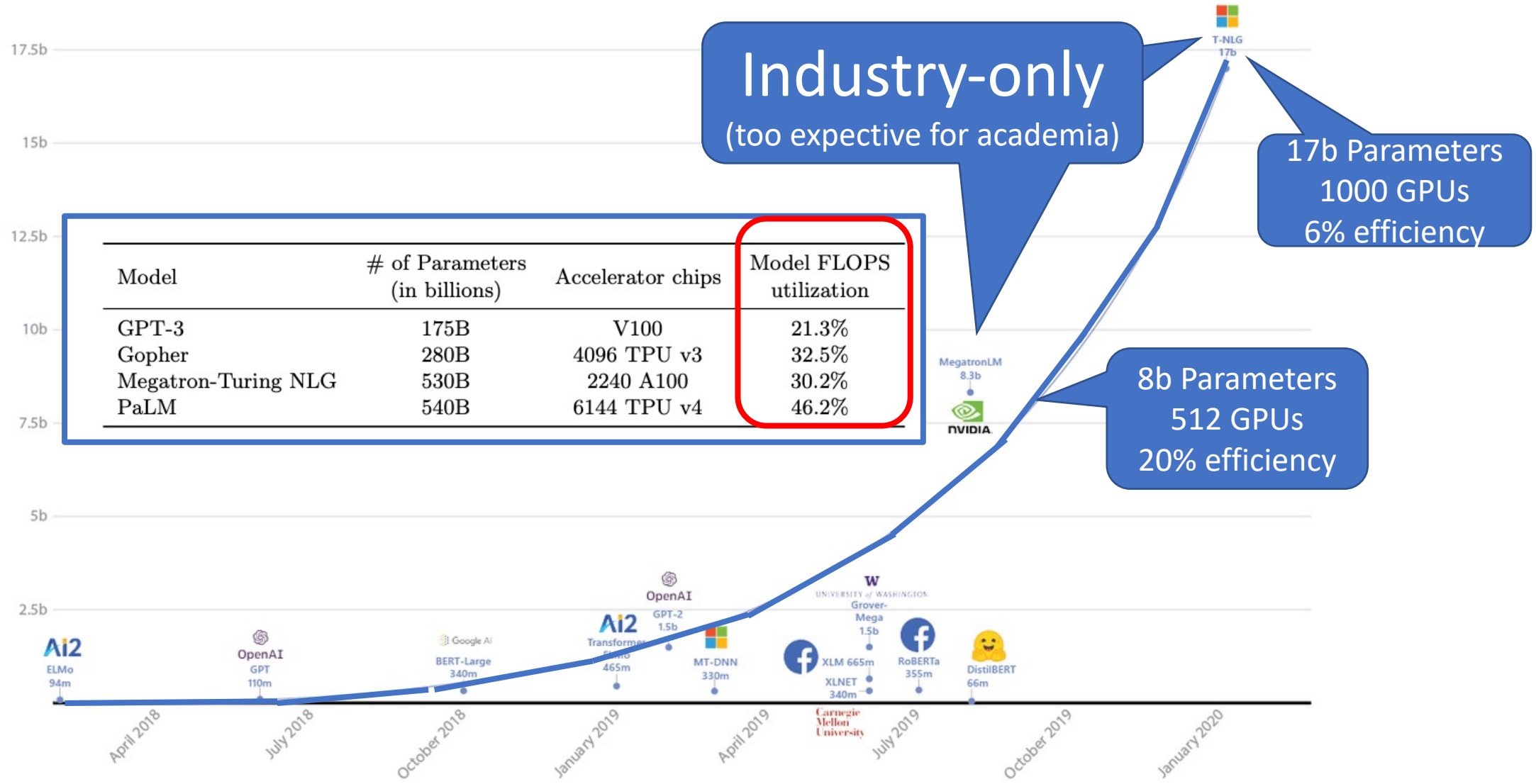
PaLM: Scaling Language Modeling with Pathways

Aakanksha Chowdhery* Sharan Narang* Jacob Devlin*
Maarten Bosma Gaurav Mishra Adam Roberts Paul Barham
Hyung Won Chung Charles Sutton Sebastian Gehrmann Parker Schuh Kensen Shi
Sasha Tsvyashchenko Joshua Maynez Abhishek Rao† Parker Barnes Yi Tay
Noam Shazeer‡ Vinodkumar Prabhakaran Emily Reif Nan Du Ben Hutchinson
Reiner Pope James Bradbury Jacob Austin Michael Isard Guy Gur-Ari
Pengcheng Yin Toju Duke Anselm Levskaya Sanjay Ghemawat Sunipa Dev
Henryk Michalewski Xavier Garcia Vedant Misra Kevin Robinson Liam Fedus
Denny Zhou Daphne Ippolito David Luan‡ Hyeontaek Lim Barret Zoph
Alexander Spiridonov Ryan Sepassi David Dohan Shivani Agrawal Mark Omernick
Andrew M. Dai Thanumalayan Sankaranarayana Pillai Marie Pellat Aitor Lewkowycz
Erica Moreira Rewon Child Oleksandr Polozov† Katherine Lee Zongwei Zhou
Xuezhi Wang Brennan Saeta Mark Diaz Orhan Firat Michele Catasta† Jason Wei
Kathy Meier-Hellstern Douglas Eck Jeff Dean Slav Petrov Noah Fiedel

Google Research

Bigger Models are Better (but less efficient)

Data Source: Microsoft blog (https://syncedreview.com/2020/02/12/17-billion-parameters-microsoft-deepspeed-breeds-worlds-largest-nlp-model/) and Kunle Olukton's presentation at ScaledML



Model	# of Parameters (in billions)	Accelerator chips	Model FLOPS utilization
GPT-3	175B	V100	21.3%
Gopher	280B	4096 TPU v3	32.5%
Megatron-Turing NLG	530B	2240 A100	30.2%
PaLM	540B	6144 TPU v4	46.2%

540B Parameters:
More Parameters → Better Scores
More Supervision → Better Scores

Task	0-shot		1-shot		Few-shot	
	Prior SOTA	PaLM 540B	Prior SOTA	PaLM 540B	Prior SOTA	PaLM 540B
TriviaQA (EM)	71.3[a]	**76.9**	75.8[a]	**81.4**	75.8[a] (1)	**81.4** (1)
Natural Questions (EM)	**24.7**[a]	21.2	26.3[a]	**29.3**	32.5[a] (1)	**39.6** (64)
Web Questions (EM)	**19.0**[a]	10.6	**25.3**[b]	22.6	41.1[b] (64)	**43.5** (64)
Lambada (EM)	77.7[f]	**77.9**	80.9[a]	**81.8**	87.2[c] (15)	**89.7** (8)
HellaSwag	80.8[f]	**83.4**	80.2[c]	**83.6**	82.4[c] (20)	**83.8** (5)
StoryCloze	83.2[b]	**84.6**	84.7[b]	**86.1**	87.7[b] (70)	**89.0** (5)
Winograd	88.3[b]	**90.1**	**89.7**[b]	87.5	88.6[a] (2)	**89.4** (5)
Winogrande	74.9[f]	**81.1**	73.7[c]	**83.7**	79.2[a] (16)	**85.1** (5)
Drop (F1)	57.3[a]	**69.4**	57.8[a]	**70.8**	58.6[a] (2)	**70.8** (1)
CoQA (F1)	**81.5**[b]	77.6	**84.0**[b]	79.9	**85.0**[b] (5)	81.5 (5)
QuAC (F1)	41.5[b]	**45.2**	43.4[b]	**47.7**	44.3[b] (5)	**47.7** (1)
SQuADv2 (F1)	71.1[a]	**80.8**	71.8[a]	**82.9**	71.8[a] (10)	**83.3** (5)
SQuADv2 (EM)	64.7[a]	**75.5**	66.5[a]	**78.7**	67.0[a] (10)	**79.6** (5)
RACE-m	64.0[a]	**68.1**	65.6[a]	**69.3**	66.9[a†] (8)	**72.1** (8)
RACE-h	47.9[c]	**49.1**	48.7[a]	**52.1**	49.3[a†] (2)	**54.6** (5)

Task	0-shot		1-shot		Few-shot	
	Prior SOTA	PaLM 540B	Prior SOTA	PaLM 540B	Prior SOTA	PaLM 540B
PIQA	82.0[c]	**82.3**	81.4[a]	**83.9**	83.2[c] (5)	**85.2** (5)
ARC-e	76.4[e]	**76.6**	76.6[a]	**85.0**	80.9[e] (10)	**88.4** (5)
ARC-c	51.4[b]	**53.0**	53.2[b]	**60.1**	52.0[a] (3)	**65.9** (5)
OpenbookQA	**57.6**[b]	53.4	**55.8**[b]	53.6	65.4[b] (100)	**68.0** (32)
BoolQ	83.7[f]	**88.0**	82.8[a]	**88.7**	84.8[c] (32)	**89.1** (8)
Copa	91.0[b]	**93.0**	**92.0**[a]	91.0	93.0[a] (16)	**95.0** (5)
RTE	**73.3**[e]	72.9	71.5[a]	**78.7**	76.8 (5)	**81.2** (5)
WiC	50.3[a]	**59.1**	52.7[a]	**63.2**	58.5[c] (32)	**64.6** (5)
Multirc (F1a)	73.7[a]	**83.5**	74.7[a]	**84.9**	77.5[a] (4)	**86.3** (5)
WSC	85.3[a]	**89.1**	83.9[a]	**86.3**	85.6[a] (2)	**89.5** (5)
ReCoRD	90.3[a]	**92.9**	90.3[a]	**92.8**	90.6 (2)	**92.9** (2)
CB	48.2[a]	**51.8**	73.2[a]	**83.9**	84.8[a] (8)	**89.3** (5)
ANLI R1	39.2[a]	**48.4**	42.4[a]	**52.6**	44.3[a] (2)	**56.9** (5)
ANLI R2	39.9[e]	**44.2**	40.0[a]	**58.7**	41.2[a] (10)	**56.1** (5)
ANLI R3	41.3[a]	**45.7**	40.8[a]	**52.3**	44.7[a] (4)	**51.2** (5)

Strengths and Opportunities

Industrial Scale SOTA-Chasing

- Strengths
  - $$
  - Staff (size of staff)
  - Logistics
- Weaknesses
  - Risk-adverse
  - Big $$ → Too Big to Fail

Opportunities for the Masses

- Smaller projects → Creativity
- Martial Arts:
  - Use their strengths to your advantage
- Community
  - Large: $|students| \gg |staff|$
  - Willingness to share: github, arXiv

Standards!!
(Builds Community)

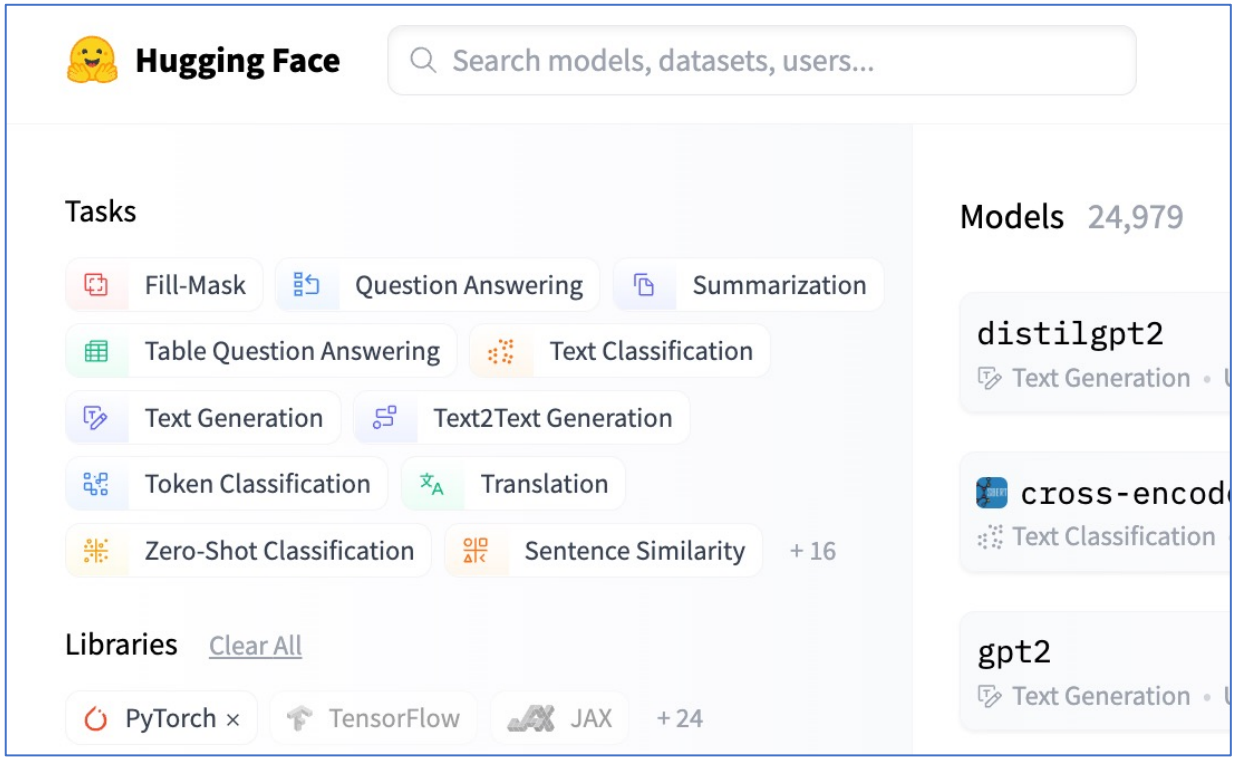
Examples:
PyTorch, gft

Hubs: PyTorch is Popular ($25k \gg 6k$)

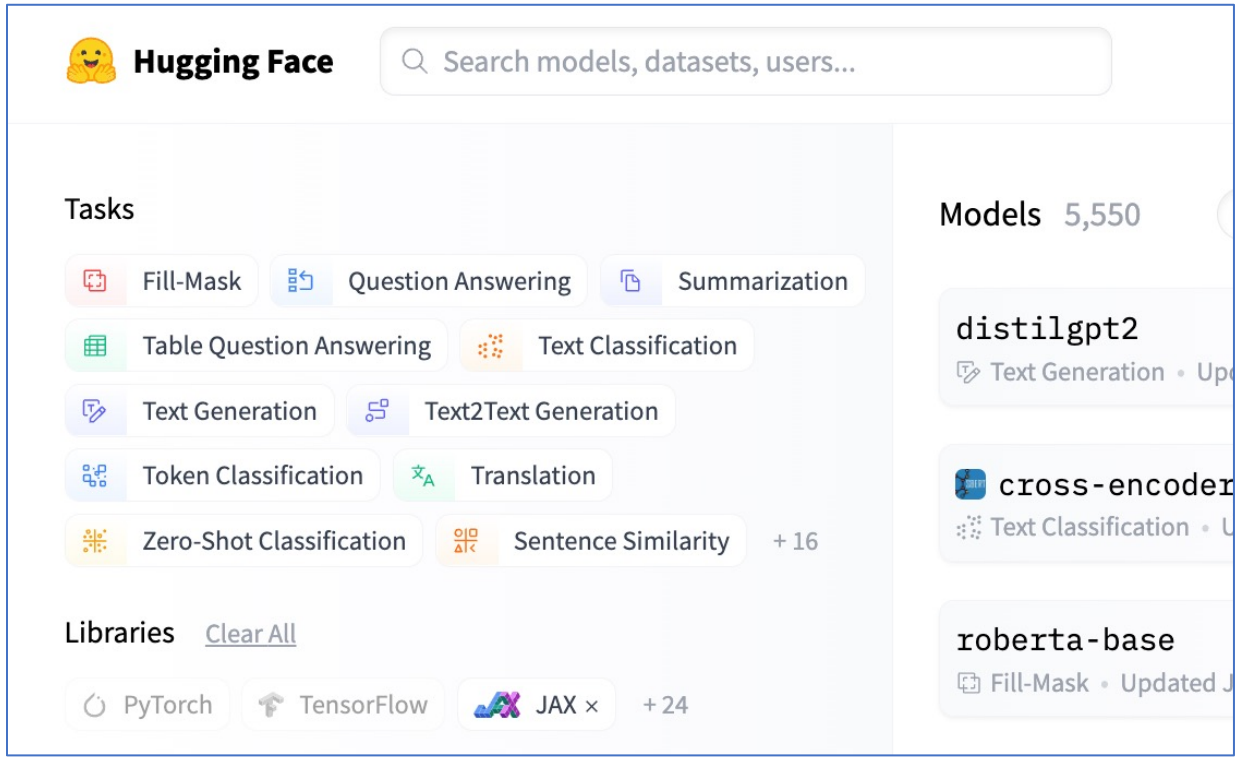
On HuggingFace, PyTorch > JAX > TensorFlow

Ease-of-Use >> System Performance

25k PyTorch Models



6k JAX Models



Why Change Terminology?

More opportunity for academics & Computational Linguistics

~~Standard Terminology~~	Proposed Alternat
~~Base/Foundation/Pre-Trained~~	f_{pre}
~~Fine-Tuning~~	*fit*
~~Inference~~	*predict*

Not this

Emphasize this

Industry has an ``unfair advantage'' over academia

- Pre-training requires $$$$, Big Data & Logistics:
- Big Investments → Risk Adverse → Less Creativity

https://www.youtube.com/watch?v=dG628PEN1fY

VLSI History

Perspective from MIT and beyond:
IBM, Silicon Valley, Mich, LGBTQ, etc.

http://www.eecs.mit.edu/docs/newsletter/VLSI.pdf
http://ai.eecs.umich.edu/people/conway/Memoirs/MIT/MIT_Reminiscences.pdf

The VLSI revolution at MIT *by Paul Penfield, Jr.*

Remember the VLSI revolution? If you are over 50, you might. If not, you have probably heard about it. In the late 1970s the VLSI (Very Large Scale Integration) revolution opened up the design of integrated circuits to people who did not know anything about device physics or semiconductor processing.

This was first demonstrated here at MIT. Below is the story of the two people, Lynn Conway (photo lower right) and Professor Jonathan Allen (photo right), who made it happen. Ideally, these two should tell the story, and Conway has written a compelling account from her perspective, "MIT Reminiscences: Student years to VLSI revolution", http://ai.eecs.umich.edu/people/conway/Memoirs/MIT/MIT_Reminiscences.pdf. Sadly, Allen died in 2000 but I will try to tell the story from the MIT EECS perspective here. Please read both accounts.

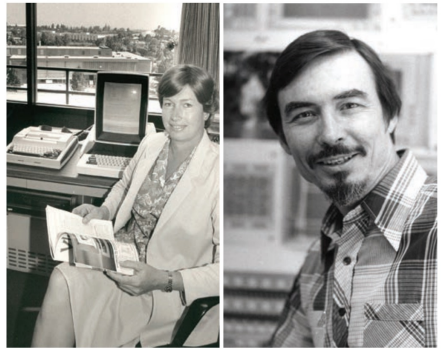
Although in the 1950s and 1960s our department (named EE at the time) was innovative in using physical device models to teach electronics, we consciously decided not to expand research in silicon devices and circuits because we thought industry could do it better.

The rise of computer science made us rethink that decision. In the early 1970s we decided not to split into separate EE and CS departments but stay united, under the new name EECS. Then we realized that our research in computer architecture and digital systems was hindered: testing new ideas in the form of hardware required LSI (large-scale integration) that was available only in industry. Allen knew this both from his own research in speech technology and through serving as Associate Director of the Research Laboratory of Electronics. He wanted fabrication as a service, not something he and other digital hardware designers would have to know how to do.

Meanwhile a revolution was brewing in California. Professor Carver Mead at Caltech had used industry connections to get student projects fabricated and the students loved it. A group centered around Lynn Conway at Xerox PARC (Palo Alto Research Center) set out to make designing integrated systems so simple that lots more people could do it. Both Stanford and Berkeley (who had continued their research in silicon devices) were interested.

In 1977 we decided we had to get on board. Several decisions followed. To keep true to our decision to stay as one department, we needed relevant research in both EE and CS. That meant: a silicon fabrication facility where research on devices and processes was informed by the needs of digital systems;

Professor Jonathan Allen was the sixth Director of the Research Laboratory of Electronics, RLE, and a member of the EECS faculty since 1968. Photo courtesy of RLE.



Lynn Conway (left) in her office at Xerox PARC (1983). On Lynn's desk are the Alto computer she used to write the VLSI textbook, and the TI terminal she used to interact with PARC while at MIT. Photo by Margaret Moulton, courtesy Lynn Conway.

- Think of f_{pre} like Intel CPU chips
- Universities can afford
  - to program CPUs and models (*fit/predict*)
  - but not VLSI fabrication (f_{pre})
- Jon Allen (my thesis advisor at MIT)
  - Invested big $$$$ in VLSI fabrication
  - Big $$$$ → (Too) Much Responsibility
    - Creativity requires willingness to take risks
- Industry can afford to work on projects
  - with industrial challenges/rewards:
    - Capital, ROI, Scale, Logistics
  - Academics have other priorities:
    - Bottom line: $fit/predict \gg f_{pre}$

Agenda: Part A

- ✓Strategy
- ✓*gft* Cheatsheet
  - ✓4-5 Functions + 4 Arguments
  - ✓Amazing how much can be done with so little
- ✓Seven Simple Examples
- ✓More Tasks: MT, ASR, Vision
- ✓Two Questions
  - ✓How do we *find* the good stuff?
  - ✓How do we *use* the good stuff?
- ✓Why de-emphasize pre-training?
- ➢**Conclusions**

Advantages of Higher-Level Little Languages

- Ease of use
  - Accessible to broader audience
- Hide complexity
  - 1-line of gft ≪ 800+ lines of python
  - More transparency; fewer bugs
- Encourage community
  - Sharing data, models, tasks
  - Replication
- Avoid special cases
  - Standard examples on hubs
    - expect users to modify python
    - different programs for different datasets/models/tasks
  - *gft* encourages code re-use
    - gft uses same code as examples
    - with same performance and computational costs
    - but generalizes over more datasets/models/tasks
- Portability across suppliers
  - Mix and Match Across Hubs
    - (HuggingFace, PaddleNLP)

Part A Conclusions

- See github for code & examples
- Amazing how much can be done with so little
- Higher level (little) languages like *gft* have many advantages over examples found on hubs:
  - Short (1-line) programs are
    - easier to read and write,
    - more transparent and
    - more portable (across hubs)
- Inclusiveness
  - Accessible to masses
- Demystify deep nets
  - No one would suggest regression-like methods are ``intelligent''
- Teaser for Part B:
  - To improve over Part A, we recommend Interdisciplinary combo
    - Deep Nets (recent progress)
    - AI Representation (decades)
    - Linguistics & Philosophy (centuries)

backup

Hundreds of Examples:
Robustness / Regression Testing

Subdirectories under $gft/examples

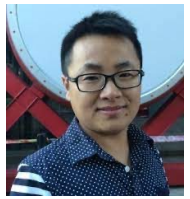
1. fit_examples
2. predict_examples
3. eval_examples
4. summary_examples

- Look for *.sh files under these directories
  - `find $gft/examples -name "*.sh"`
- 4 arguments:
  - --data, --model, --eqn, --task

4 Functions

1. gft_fit: $f_{pre} \rightarrow f_{post}$ (fine-tuning)
   - 4 Arguments, --output_dir, --metric, --splits
   - (plus most args in most hubs)
2. gft_predict: $f(x) \rightarrow \hat{y}$ (inference)
   - Input: 4 Arguments (x from data or stdin)
   - Output: $\hat{y}$ for each x
3. gft_eval: Score model on dataset
   - Input: 4 Arguments, --metric, …
   - Output: Score
4. gft_summary: Find good stuff
   - 4 Arguments
   - (may include: __*contains*__, __*infer*__)

Part A: Simple Uses of Deep Nets



Yanjun

Zeyu

Schedule

- Overview (15 min)
- Part A: (60 min)
  - Glass is half full
- Break/Q&A (10 min)
- Part B: (60 min)
  - Glass is half empty
  - There is always more work to do
- Conclusions (15 min)
- Break/Q&A (10 min)

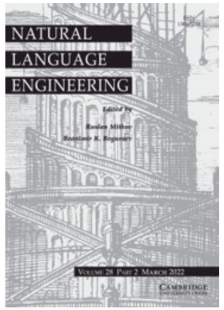
Valia

Gary

Ernie

Part A: Glass is Half Full

Simple/Popular Methods are Powerful



Natural Language Engineering

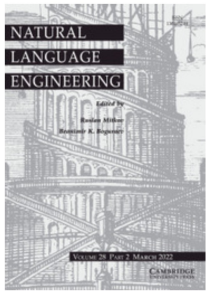
Emerging trends: Deep nets for poets

Published online by Cambridge University Press: **01 September 2021**

Kenneth Ward Church, Xiaopeng Yuan, Sheng Guo, Zewu Wu, Yehua Yang and Zeyu Chen

Article | Figures | Metrics

Save PDF | Share | Cite | Rights & Permissions



Natural Language Engineering

Emerging trends: A gentle introduction to fine-tuning

Published online by Cambridge University Press: **26 October 2021**

Kenneth Ward Church, Zeyu Chen and Yanjun Ma

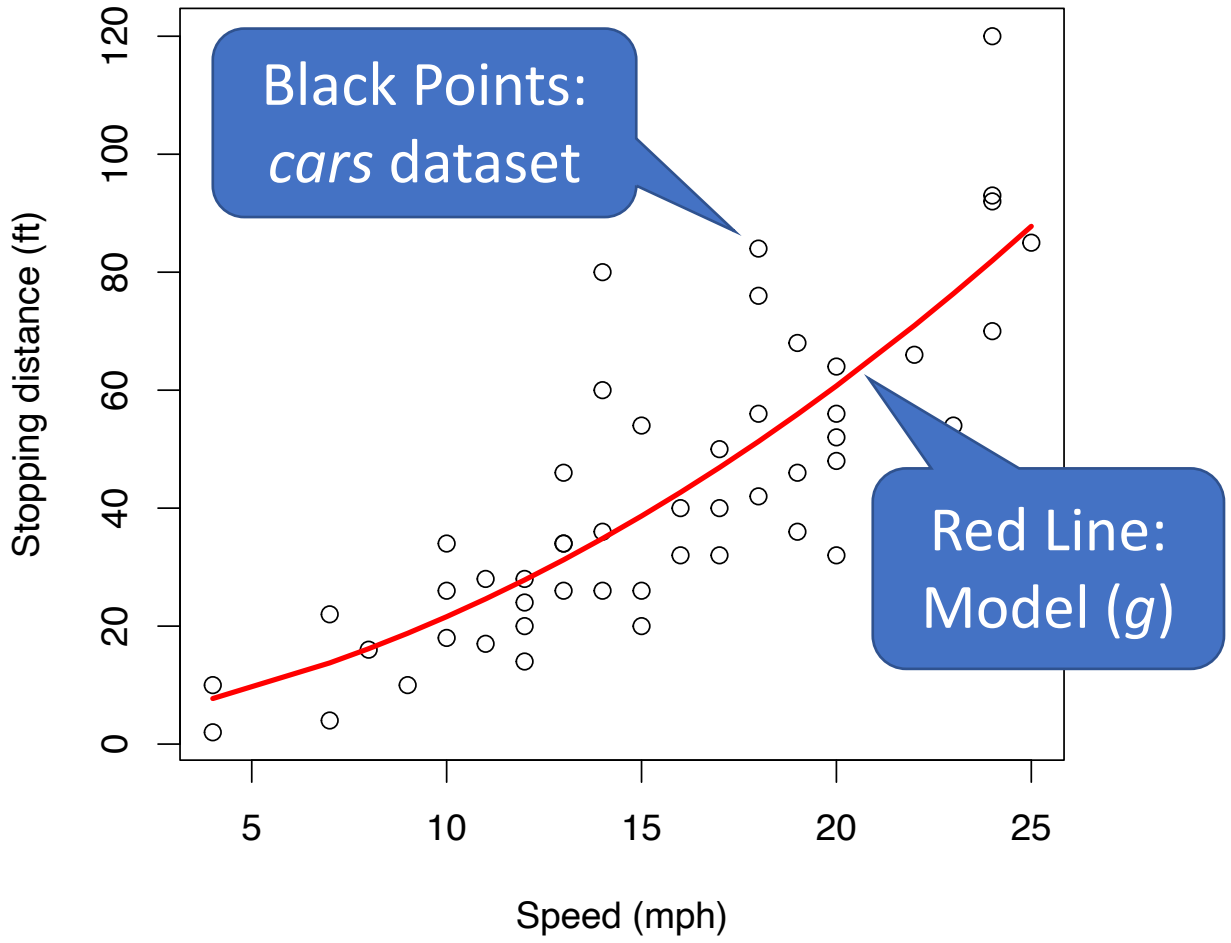
Show author details

Article | Figures | Metrics

Save PDF | Share | Cite | Rights & Permissions

Regression in *R*:

Summarize (almost) anything




```
> head(cars)
  speed dist
1     4    2
2     4   10
3     7    4
4     7   22
5     8   16
6     9   10
> summary(cars)
     speed           dist       
 Min.   : 4.0   Min.   :  2.00  
 1st Qu.:12.0   1st Qu.: 26.00  
 Median :15.0   Median : 36.00  
 Mean   :15.4   Mean   : 42.98  
 3rd Qu.:19.0   3rd Qu.: 56.00  
 Max.   :25.0   Max.   :120.00  
>
```

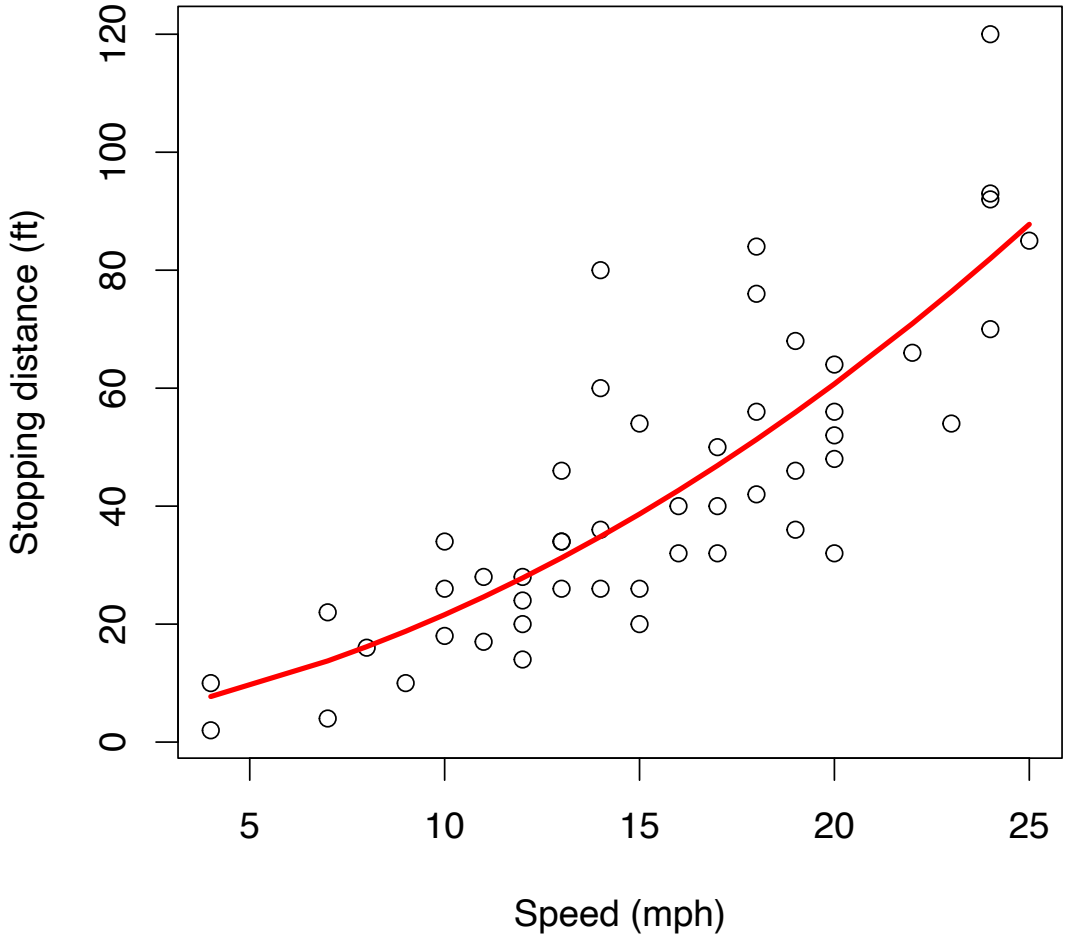
Summarize a dataset (*cars*)

Fit a model (*g*) to data (*cars*)

Regression in *R*:

Summarize (almost) anything

```
> plot(cars, xlab="Speed (mph)", ylab="Stopping distance (ft)")
> g = glm(dist ~ poly(speed,2), data=cars)
> o = order(cars$speed)
> lines(cars$speed[o], predict(g,cars)[o], col="red", lwd=3)
```



Regression in *R*:

Summarize (almost) anything

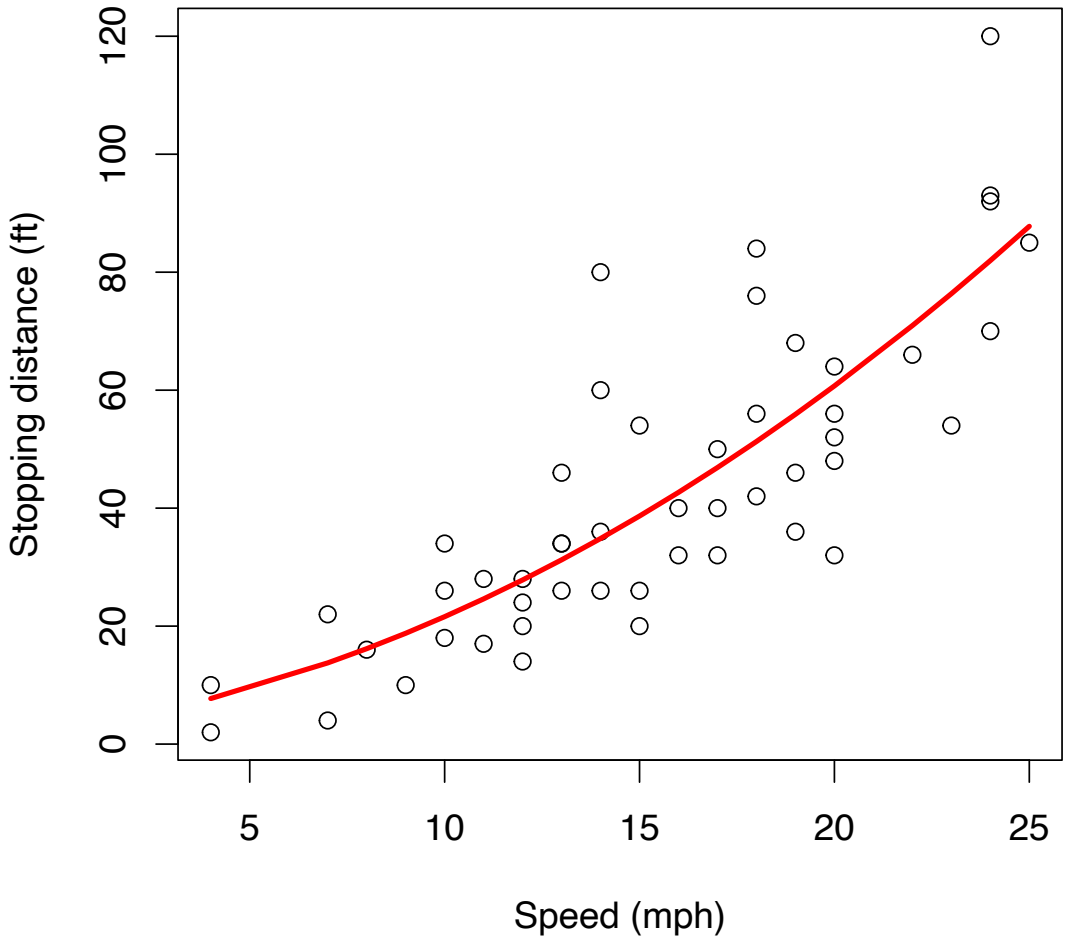
Fit a model (*g*) to data (*cars*)

```
> plot(cars, xlab="Speed (mph)", ylab="Stopping distance (ft)")
> g = glm(dist ~ poly(speed,2), data=cars)
> o = order(cars$speed)
> lines(cars$speed[o], predict(g,cars)[o], col="red", lwd=3)
> summary(g)
```

Summarize a model (*g*)

```
Call:
glm(formula = dist ~ poly(speed, 2), data = cars)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-28.720   -9.184   -3.188    4.628   45.152

Coefficients:
                Estimate Std. Error t value Pr(>|t|)
(Intercept)       42.980      2.146  20.026  < 2e-16 ***
poly(speed, 2)1  145.552     15.176   9.591 1.21e-12 ***
poly(speed, 2)2   22.996     15.176   1.515    0.136
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for gaussian family taken to be 230.3131)

    Null deviance: 32539  on 49  degrees of freedom
Residual deviance: 10825  on 47  degrees of freedom
AIC: 418.77

Number of Fisher Scoring iterations: 2
```

Opportunity to Improve *g*



gft_summary:
Summarize (almost) anything: datasets, models, tasks, etc.

```
# summarize the qqp subtask of glue (from HuggingFace)
gft_summary --data H:glue,qqp 2>/dev/null
# dataset: glue,qqp splits: train: 363846 rows, test: 390965 rows, val: 40430 rows
# dataset: glue,qqp split: train    columns: question1, question2, label, idx
# dataset: glue,qqp labels: not_duplicate, duplicate
```

How do I find the good stuff? Popular Models? gft_summary –model H:__contains__snli --topn 20

Model	Downloads	Likes	Task
ynie/roberta-large-snli_mnli_fever_anli_R1_R2_R3-nli	31,354	2	text-classification
textattack/bert-base-uncased-snli	6,651	0	text-classification
boychaboy/SNLI_roberta-base	3,158	0	text-classification
symanto/sn-xlm-roberta-base-snli-mnli-anli-xnli	2,575	1	sentence-similarity
ynie/albert-xxlarge-v2-snli_mnli_fever_anli_R1_R2_R3-nli	2,093	1	text-classification
pritamdeka/S-Biomed-Roberta-snli-multinli-stsb	671	0	sentence-similarity
jegorkitskerkin/bert-base-dutch-cased-snli	528	1	sentence-similarity
ynie/bart-large-snli_mnli_fever_anli_R1_R2_R3-nli	274	0	text-classification
symanto/xlm-roberta-base-snli-mnli-anli-xnli	151	1	text-classification
persiannlp/mt5-base-parsinlu-snli-entailment	115	0	text2text-generation
usc-isi/sbert-roberta-large-anli-mnli-snli	97	0	sentence-similarity
ynie/xlnet-large-cased-snli_mnli_fever_anli_R1_R2_R3-nli	94	0	text-classification
textattack/albert-base-v2-snli	67	0	text-classification
pmthangk09/bert-base-uncased-esnli	53	0	text-classification
boychaboy/SNLI_bert-base-uncased	46	0	text-classification
ynie/electra-large-discriminator-snli_mnli_fever_anli_R1_R2_R3-nli	44	0	text-classification
NDugar/debertav3-mnli-snli-anli	42	2	zero-shot-classification
boychaboy/SNLI_bert-large-cased	39	0	text-classification
persiannlp/mt5-small-parsinlu-snli-entailment	37	0	text2text-generation
boychaboy/SNLI_roberta-large	33	0	text-classification

Embarrassment of Riches

✓Hubs have 30k models & 3k datasets (increasing 3x per year)

- Q: How can I find the good stuff?

➢ **A: *gft_summary***

- ➢ __contains__
  - ➢ Find popular models/datasets that contain substring
- ➢ __infer__
  - ➢ Find popular models that are associated with task/dataset

➢ Examples:

- ➢ gft_summary --data H:__contains__emotion --topn 5
- ➢ gft_summary --model H:__contains__emotion --topn 5
- ➢ gft_summary --task H:classify --model H:__infer__ --topn 5
- ➢ gft_summary --data H:emotion --model H:__infer__ --topn 5

What is ``Good'' Stuff? Accuracy? Downloads?

Hyper-Parameter Tuning → Small Gains

Best of Hubs

Model	VAcc	D
C:RoBERTa large, tuned by Yuchen Bian	0.924	
H:textattack/roberta-base-MRPC	0.912	1623
H:textattack/albert-base-v2-MRPC	0.897	175
H:mrm8488/deberta-v3-small-finetuned-mrpc	0.892	30
H:textattack/bert-base-uncased-MRPC	0.877	10,133
H:textattack/distilbert-base-uncased-MRPC	0.858	108
H:ajrae/bert-base-uncased-finetuned-mrpc	0.858	115
C:*gft_fit* example (BERT with no tuning)	0.853	
H:textattack/distilbert-base-cased-MRPC	0.784	122

Table 4: *gft* achieves VAcc (accuracy on validation split) close to distilbert (compressed) models. HuggingFace models were selected using *gft_summary* to find popular models by downloads (D).

SOTA

Source	Test Accuracy
GLUE Leaderboard (L)	0.945
Papers with code (PWC)	0.937
Human Baseline (HB)	0.863

Table 5: SOTA (state-of-the-art) for MRPC (GLUE).

Should not compare accuracy across splits

Best of Hubs are good (but probably not SOTA)

Popular Text Classification Models
gft_summary --task H:text-classification
Input: Task → Output: Popular Models

Model	Downloads	Likes
cross-encoder/ms-marco-MiniLM-L-12-v2	11,542,439	4
distilbert-base-uncased-finetuned-sst-2-english	4,188,846	38
facebook/bart-large-mnli	1,253,261	89
cross-encoder/nli-distilroberta-base	581,821	6
cardiffnlp/twitter-roberta-base-sentiment	542,530	40
nlptown/bert-base-multilingual-uncased-sentiment	541,492	30
roberta-large-mnli	520,103	12
finiteautomata/beto-sentiment-analysis	496,725	6
j-hartmann/emotion-english-distilroberta-base	273,134	16
cardiffnlp/twitter-roberta-base-emotion	259,369	5

Summarize Tasks $gft/examples/summary_examples/task

- gft_summary --task H:audio-classification
- gft_summary --task H:audio-to-audio
- gft_summary --task H:automatic-speech-recognition
- gft_summary --task H:conversational
- gft_summary --task H:feature-extraction
- gft_summary --task H:fill-mask
- gft_summary --task H:image-classification
- gft_summary --task H:image-segmentation
- gft_summary --task H:image-to-text
- gft_summary --task H:object-detection
- gft_summary --task H:protein-folding
- gft_summary --task H:question-answering
- gft_summary --task H:sentence-similarity
- gft_summary --task H:speech-segmentation
- gft_summary --task H:structured-data-classification
- gft_summary --task H:summarization
- gft_summary --task H:table-question-answering
- gft_summary --task H:text2text-generation
- **gft_summary --task H:text-classification**
- gft_summary --task H:text-generation
- gft_summary --task H:text-to-image
- gft_summary --task H:text-to-speech
- gft_summary --task H:token-classification
- gft_summary --task H:translation
- gft_summary --task H:voice-activity-detection
- gft_summary --task H:zero-shot-classification
- gft_summary --task H:zero-shot-image-classification

Specifying –model argument

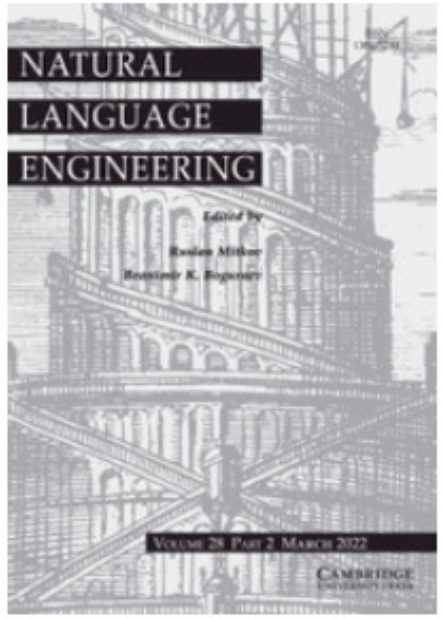
```
# text-classification: sentiment analysis
echo 'I love you.' | gft_predict --task text-classification
# I love you.   POSITIVE    0.9998705387115479

# text-classification: emotion classification
model=H:AdapterHub/bert-base-uncased-pf-emotion
echo 'I love you.' | gft_predict --model $model --task text-classification
# I love you.   love    0.6005669236183167
```

Default Model → Sentiment Classes

A Model with Emotion Classes

SOTA-Chasing: Leaderboards Considered Harmful

SOTA-chasing refers to papers that report SOTA numbers, but contribute little of lasting value to the literature. The point is the pointlessness.



Natural Language Engineering

Emerging Trends: SOTA-Chasing

Published online by Cambridge University Press: **08 February 2022**

Kenneth Ward Church and Valia Kordoni

Article | Figures | Metrics

Save PDF | Share | Cite

Abstract

Most read

This page lists the top ten most read articles for this journal based on the number of full text views and downloads recorded on Cambridge Core over the last 30 days. This list is updated on a daily basis.

- **GPT-3: What's it good for?** — 13
  Robert Dale
  Published online by Cambridge University Press: 15 December 2020, pp. 113-118
  Article | Access | Open access | PDF | HTML | Export citation
  View abstract

- **Word2Vec** — 10
  KENNETH WARD CHURCH
  Published online by Cambridge University Press: 16 December 2016, pp. 155-162
  Article | Access | Open access | PDF | HTML | Export citation
  View abstract

- **Emerging Trends: SOTA-Chasing** — 17
  Kenneth Ward Church, Valia Kordoni
  Published online by Cambridge University Press: 08 February 2022, pp. 249-269
  Article | Access | Open access | PDF | HTML | Export citation
  View abstract

PaLM (540B Parameters):

Bigger is Better; Supervised is Better

Compressed f_{pre} → Smaller

Task	0-shot		1-shot		Few-shot	
	Prior SOTA	PaLM 540B	Prior SOTA	PaLM 540B	Prior SOTA	PaLM 540B
TriviaQA (EM)	71.3[a]	**76.9**	75.8[a]	**81.4**	75.8[a] (1)	**81.4** (1)
Natural Questions (EM)	**24.7**[a]	21.2	26.3[a]	**29.3**	32.5[a] (1)	**39.6** (64)
Web Questions (EM)	**19.0**[a]	10.6	**25.3**[b]	22.6	41.1[b] (64)	**43.5** (64)
Lambada (EM)	77.7[f]	**77.9**	80.9[a]	**81.8**	87.2[c] (15)	**89.7** (8)
HellaSwag	80.8[f]	**83.4**	80.2[c]	**83.6**	82.4[c] (20)	**83.8** (5)
StoryCloze	83.2[b]	**84.6**	84.7[b]	**86.1**	87.7[b] (70)	**89.0** (5)
Winograd	88.3[b]	**90.1**	**89.7**[b]	87.5	88.6[a] (2)	**89.4** (5)
Winogrande	74.9[f]	**81.1**	73.7[c]	**83.7**	79.2[a] (16)	**85.1** (5)
Drop (F1)	57.3[a]	**69.4**	57.8[a]	**70.8**	58.6[a] (2)	**70.8** (1)
CoQA (F1)	**81.5**[b]	77.6	**84.0**[b]	79.9	**85.0**[b] (5)	81.5 (5)
QuAC (F1)	41.5[b]	**45.2**	43.4[b]	**47.7**	44.3[b] (5)	**47.7** (1)
SQuADv2 (F1)	71.1[a]	**80.8**	71.8[a]	**82.9**	71.8[a] (10)	**83.3** (5)
SQuADv2 (EM)	64.7[a]	**75.5**	66.5[a]	**78.7**	67.0[a] (10)	**79.6** (5)
RACE-m	64.0[a]	**68.1**	65.6[a]	**69.3**	66.9[a†] (8)	**72.1** (8)
RACE-h	47.9[c]	**49.1**	48.7[a]	**52.1**	49.3[a†] (2)	**54.6** (5)

Task	SOTA	540B	SOTA	540B	SOTA	PaLM 540B
PIQA	82.0[c]	**82.3**	81.4[a]	**83.9**	83.2[c] (5)	**85.2** (5)
ARC-e	76.4[e]	**76.6**	76.6[a]	**85.0**	80.9[e] (10)	**88.4** (5)
ARC-c	51.4[b]	**53.0**	53.2[b]	**60.1**	52.0[a] (3)	**65.9** (5)
OpenbookQA	**57.6**[b]	53.4	**55.8**[b]	53.6	65.4[b] (100)	**68.0** (32)
BoolQ	83.7[f]	**88.0**	82.8[a]	**88.7**	84.8[c] (32)	**89.1** (8)
Copa	91.0[b]	**93.0**	**92.0**[a]	91.0	93.0[a] (16)	**95.0** (5)
RTE	**73.3**[e]	72.9	71.5[a]	**78.7**	76.8 (5)	**81.2** (5)
WiC	50.3[a]	**59.1**	52.7[a]	**63.2**	58.5[c] (32)	**64.6** (5)
Multirc (F1a)	73.7[a]	**83.5**	74.7[a]	**84.9**	77.5[a] (4)	**86.3** (5)
WSC	85.3[a]	**89.1**	83.9[a]	**86.3**	85.6[a] (2)	**89.5** (5)
ReCoRD	90.3[a]	**92.9**	90.3[a]	**92.8**	90.6 (2)	**92.9** (2)
CB	48.2[a]	**51.8**	73.2[a]	**83.9**	84.8[a] (8)	**89.3** (5)
ANLI R1	39.2[a]	**48.4**	42.4[a]	**52.6**	44.3[a] (2)	**56.9** (5)
ANLI R2	39.9[e]	**44.2**	40.0[a]	**58.7**	41.2[a] (10)	**56.1** (5)
ANLI R3	41.3[a]	**45.7**	40.8[a]	**52.3**	44.7[a] (4)	**51.2** (5)

Pain Points & Opportunities

Standard Terminology	Proposed Alternatives
~~Base/Foundation/Pre-Trained~~	f_{pre}
~~Fine-Tuning~~	*fit*
~~Inference~~	*predict*

One-time Cost

Recurring Costs