Telefónica

Recent advances in applied chatbot technology

to be presented at AI Ukraine 2017

Jordi Carrera Ventura

NLP scientist
Telefónica Research & Development

Outline

Overview

Industrial state-of-the-art

Current challenges

Recent advances

Conclusions

Overview

User> *what are you?*

Chatbots are a form of conversational artificial intelligence (AI) in which a user interacts with a virtual agent through natural language messaging in
- a messaging interface like Slack or Facebook Messenger
- or a voice interface like Amazon Echo or Google Assistant.

Chatbot[1]
A bot that lives in a chat (an automation routine inside a UI).

Conversational agent

Virtual Assistant[1]
A bot that takes natural language as input and returns it as output.

Chatbot[2]
A virtual assistant that lives in a chat.

[1] Generally assumed to have broader coverage and more advanced AI than chatbots[2].

Use cases

Usually intended to
- get quick answers to a specific questions over some pre-defined repository of knowledge
- perform transactions in a faster and more natural way.

Can be used to
- surface contextually relevant information
- help a user complete an online transaction
- serve as a helpdesk agent to resolve a customer’s issue without ever involving a human.

Virtual assistants for
Customer support, e-commerce, expense management, booking flights, booking meetings, data science...

What's really important...

Conversational technology should really be about

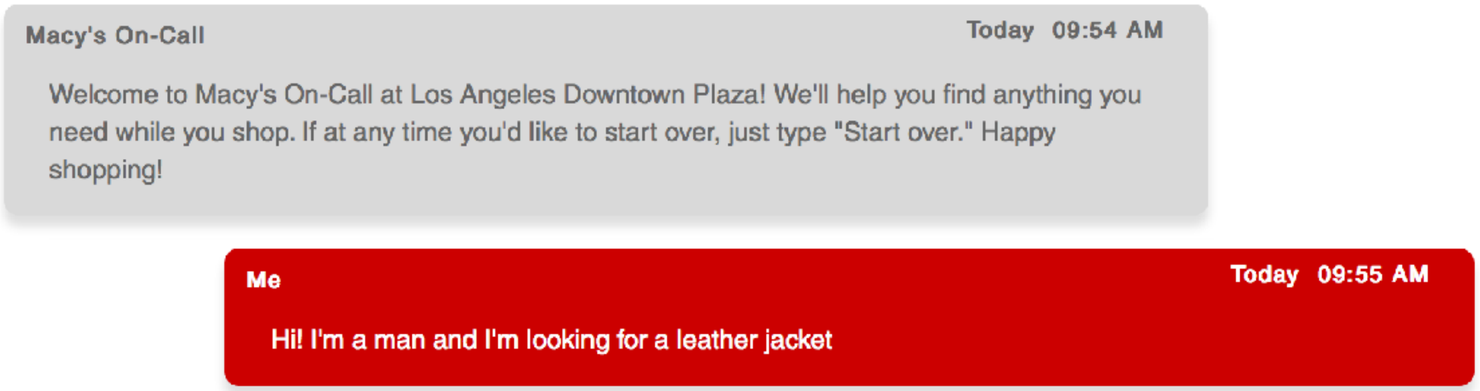
- dynamically finding the best possible way to browse a large repository of information/actions.
- find the shortest path to any relevant action or piece of information (to avoid the *plane dashboard* effect).

What's really important...

Conversational technology should really be about

- dynamically finding the best possible way to browse a large repository of information/actions.
- find the shortest path to any relevant action or piece of information (to avoid the *plane dashboard* effect).
- surfacing implicit data in unstructured content ("*bocadillo de calamares in Madrid*"). Rather than going open-domain, taking the closed-domain and going deep into it.

... and the hype


Macy's On-Call
Today 09:54 AM
Welcome to Macy's On-Call at Los Angeles Downtown Plaza! We'll help you find anything you need while you shop. If at any time you'd like to start over, just type "Start over." Happy shopping!
Me
Today 09:55 AM
Hi! I'm a man and I'm looking for a leather jacket

... and the hype

Macy's On-Call Today 09:54 AM

Welcome to Macy's On-Call at Los Angeles Downtown Plaza! We'll help you find anything you need while you shop. If at any time you'd like to start over, just type "Start over." Happy shopping!

Me Today 09:55 AM

Hi! I'm a man and I'm looking for a leather jacket

Macy's On-Call Today 09:55 AM

We've got what you're looking for!
Head to Floor 1 near Men's Collections
For more info on colors, sizes & more, click the image below.

Michael Michael Kors Men's Lea...

... and the hype


Me
Today 09:56 AM
that's a nice one but what if I want to browse different jackets? is there an easy way of doing it?

... and the hype

Me Today 09:56 AM

that's a nice one but what if I want to browse different jackets? is there an easy way of doing it?

Macy's On-Call Today 09:56 AM

We've got what you're looking for!
Head to Floor 1 near Men's Collections
For more info on colors, sizes & more, click the image below.

Marc New York Men's Leather Mo...

... and the hype


Me
Today 09:57 AM
where can I find a pair of shoes that match this jacket?

... and the hype

Me Today 09:57 AM

where can I find a pair of shoes that match this jacket?

Macy's On-Call Today 09:57 AM

We've got what you're looking for!
Head to Floor 1 near Men's Collections
For more info on colors, sizes & more, click the image below.

Michael Kors Men's Leather Rac...

... and the hype

... and the hype

Me Today 09:57 AM

pair of shoes

Macy's On-Call Today 09:57 AM

We've got what you're looking for!
Head to Floor 1 near Men's Collections
For more info on colors, sizes & more, click the image below.

Rockport Style Leader 2 Whitne...

... and the hype

That was IBM Watson 🙄

Industrial state-of-the-art

Industrial state-of-the-art

Lack of suitable metrics

	Twitter				Ubuntu			
Metric	**Spearman**	p-value	**Pearson**	p-value	**Spearman**	p-value	**Pearson**	p-value
Greedy	0.2119	0.034	0.1994	0.047	0.05276	0.6	0.02049	0.84
Average	0.2259	0.024	0.1971	0.049	-0.1387	0.17	-0.1631	0.10
Extrema	0.2103	0.036	0.1842	0.067	0.09243	0.36	-0.002903	0.98
METEOR	0.1887	0.06	0.1927	0.055	0.06314	0.53	0.1419	0.16
BLEU-1	0.1665	0.098	0.1288	0.2	-0.02552	0.8	0.01929	0.85
BLEU-2	0.3576	< 0.01	0.3874	< 0.01	0.03819	0.71	0.0586	0.56
BLEU-3	0.3423	< 0.01	0.1443	0.15	0.0878	0.38	0.1116	0.27
BLEU-4	0.3417	< 0.01	0.1392	0.17	0.1218	0.23	0.1132	0.26
ROUGE	0.1235	0.22	0.09714	0.34	0.05405	0.5933	0.06401	0.53
Human	0.9476	< 0.01	1.0	0.0	0.9550	< 0.01	1.0	0.0

Table 3: Correlation between each metric and human judgements for each response. Correlations shown in the human row result from randomly dividing human judges into two groups.

Liu, Chia-Wei, et al. "How NOT to evaluate your dialogue system: An empirical study of unsupervised evaluation metrics for dialogue response generation." arXiv preprint arXiv: 1603.08023 (2016).

Industrial state-of-the-art

Benchmark

By Intento (https://inten.to)

Dataset
SNIPS.ai 2017 NLU Benchmark
https://github.com/snipsco/nlu-benchmark

Example intents

SearchCreativeWork (e.g. *Find me the I, Robot television show*)
GetWeather (e.g. *Is it windy in Boston, MA right now?)*
BookRestaurant (e.g. *I want to book a highly rated restaurant for me and my boyfriend tomorrow night)*
PlayMusic (e.g. *Play the last track from Beyoncé off Spotify)*
AddToPlaylist (e.g. *Add Diamonds to my roadtrip playlist)*
RateBook (e.g. *Give 6 stars to Of Mice and Men)*
SearchScreeningEvent (e.g. *Check the showtimes for Wonder Woman in Paris)*

Industrial state-of-the-art

Benchmark

Methodology

English language.

Removed duplicates that differ by number of whitespaces, quotes, lettercase, etc.

Resulting dataset parameters
7 intents, 15.6K utterances (~2K per intent)

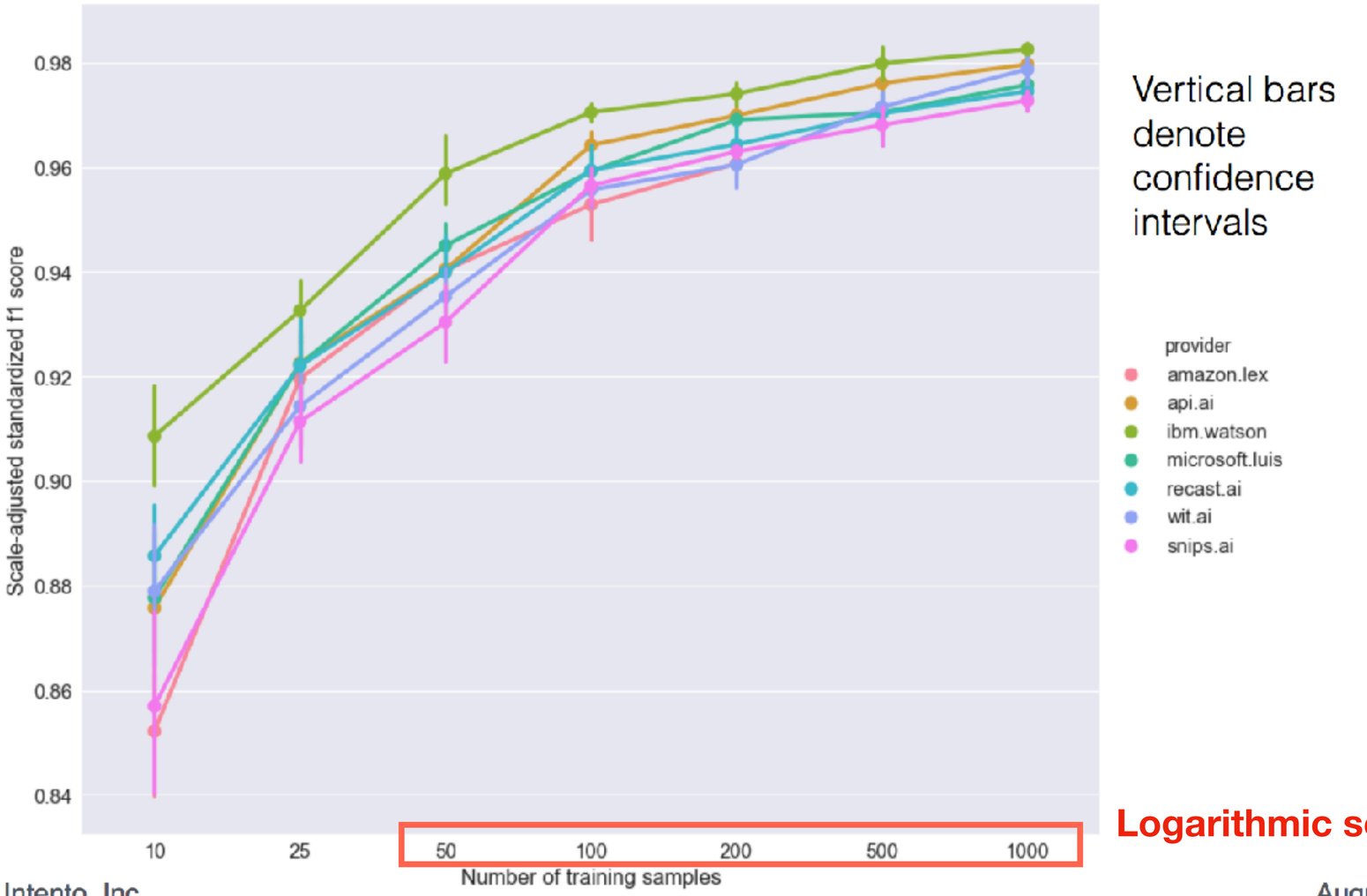
3-fold 80/20 cross-validation.

Most providers do not offer programmatic interfaces for adding training data.

Industrial state-of-the-art

Framework	Since	F1	False positives	Response time	Price
IBM Watson Conversation	2015	99.7	100%	0.35	2.5
API.ai (Google 2016)	2010	99.6	40%	0.28	Free
Microsoft LUIS	2015	99.2	100%	0.21	0.75
Amazon Lex	2016	96.5	82%	0.43	0.75
Recast.ai	2016	97	75%	2.06	N/A
wit.ai (Facebook)	2013	97.4	72%	0.96	Free
SNIPS	2017	97.5	26%	0.36	Per device

Leaning curve by provider



Logarithmic scale!

Current challenges

So, let's be real...

Welcome to Macy's blah-blah... !

hi im a guy looking for leda jackets

We've got what you're looking for!
<image product_id=1 item_id=1>

that's a nice one but, i want to see different jackets .... how can i do that

So, let's be real...

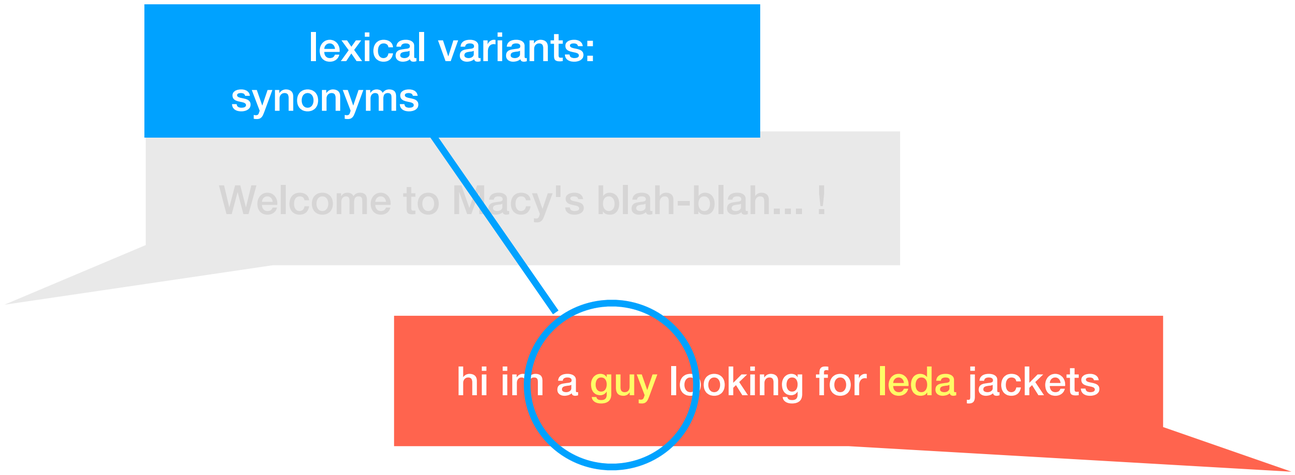
We've got what you're looking for!
<image product_id=1 item_id=2>

where can i find a pair of shoes that match this yak etc.

We've got what you're looking for!
<image product_id=1 item_id=3>

pair of shoes
(, you talking dishwasher)

So, let's be real...

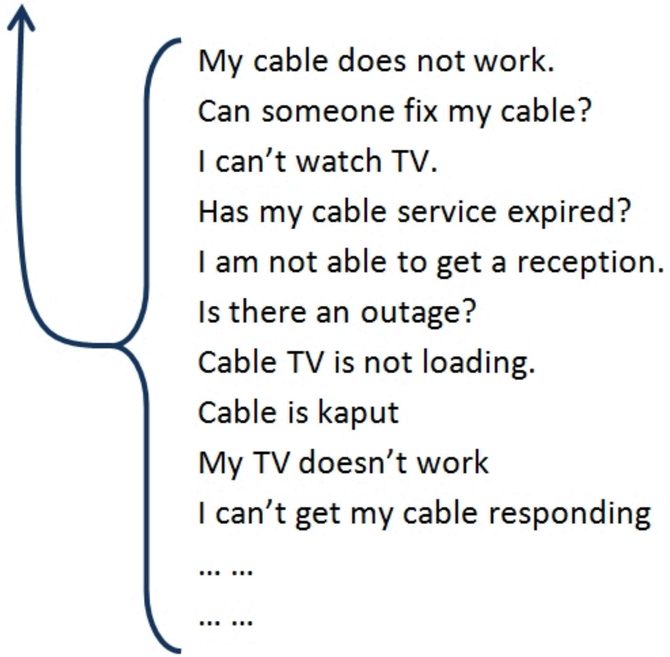


So, let's be real...

lexical variants:
synonyms, paraphrases

Intent

I have a problem with my cable.



My cable does not work.
Can someone fix my cable?
I can’t watch TV.
Has my cable service expired?
I am not able to get a reception.
Is there an outage?
Cable TV is not loading.
Cable is kaput
My TV doesn’t work
I can’t get my cable responding
… …
… …

So, let's be real...

lexical variants:
synonyms, paraphrases

Entity

Intent

I have a problem with my cable.

My cable does not work.
Can someone fix my cable?
I can't watch TV.
Has my cable service expired?
I am not able to get a reception.
Is there an outage?
Cable TV is not loading.
Cable is kaput
My TV doesn't work
I can't get my cable responding
… …
… …

My cable does not work.
Can someone fix my cable?
I can't watch TV.
Has my cable service expired?
I am not able to get a reception.
Is there an outage?
Cable TV is not loading.
Cable is kaput
My TV doesn't work
I can't get my cable responding
… …
… …

I am not able to watch TV
I am not able to watch television
I am not able to watch cable
I am not able to watch cable TV
I cannot watch TV
I cannot watch television
I cannot watch cable
I cannot watch cable TV
Cannot watch TV
Cannot watch television
Cannot watch cable
Cannot watch cable TV
Can't watch TV
Can't watch television
Can't watch cable
Can't watch cable TV

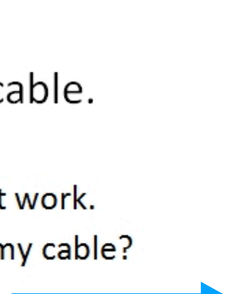
So, let's be real...

lexical variants:
synonyms, paraphrases

Intent

I have a problem with my cable.

My cable does not work.
Can someone fix my cable?
I can't watch TV.
Has my cable service expired?
I am not able to get a reception.
Is there an outage?
Cable TV is not loading.
Cable is kaput
My TV doesn't work
I can't get my cable responding
… …
… …



My cable does not work.
Can someone fix my cable?
I can't watch TV.
Has my cable service expired?
I am not able to get a reception.
Is there an outage?
Cable TV is not loading.
Cable is kaput
My TV doesn't work
I can't get my cable responding
… …
… …

Entity

I am not able to watch TV
I am not able to watch television
I am not able to watch cable
I am not able to watch cable TV
I cannot watch TV
I cannot watch television
I cannot watch cable
I cannot watch cable TV
Cannot watch TV
Cannot watch television
Cannot watch cable
Cannot watch cable TV
Can't watch TV
Can't watch television
Can't watch cable
Can't watch cable TV

This can be normalized, e.g. regular expressions

So, let's be real...

lexical variants:
synonyms, paraphrases

Intent

I have a problem with my cable.

My cable does not work.
Can someone fix my cable?
I can't watch TV.
Has my cable service expired?
I am not able to get a reception.
Is there an outage?
Cable TV is not loading.
Cable is kaput
My TV doesn't work
I can't get my cable responding
… …
… …

My cable does not work.
Can someone fix my cable?
I can't watch TV.
Has my cable service expired?
I am not able to get a reception.
Is there an outage?
Cable TV is not loading.
Cable is kaput
My TV doesn't work
I can't get my cable responding
… …
… …

Entity

I am not able to watch TV
I am not able to watch television
I am not able to watch cable
I am not able to watch cable TV
I cannot watch TV
I cannot watch television
I cannot watch cable
I cannot watch cable TV
Cannot watch TV
Cannot watch television
Cannot watch cable
Cannot watch cable TV
Can't watch TV
Can't watch television
Can't watch cable
Can't watch cable TV

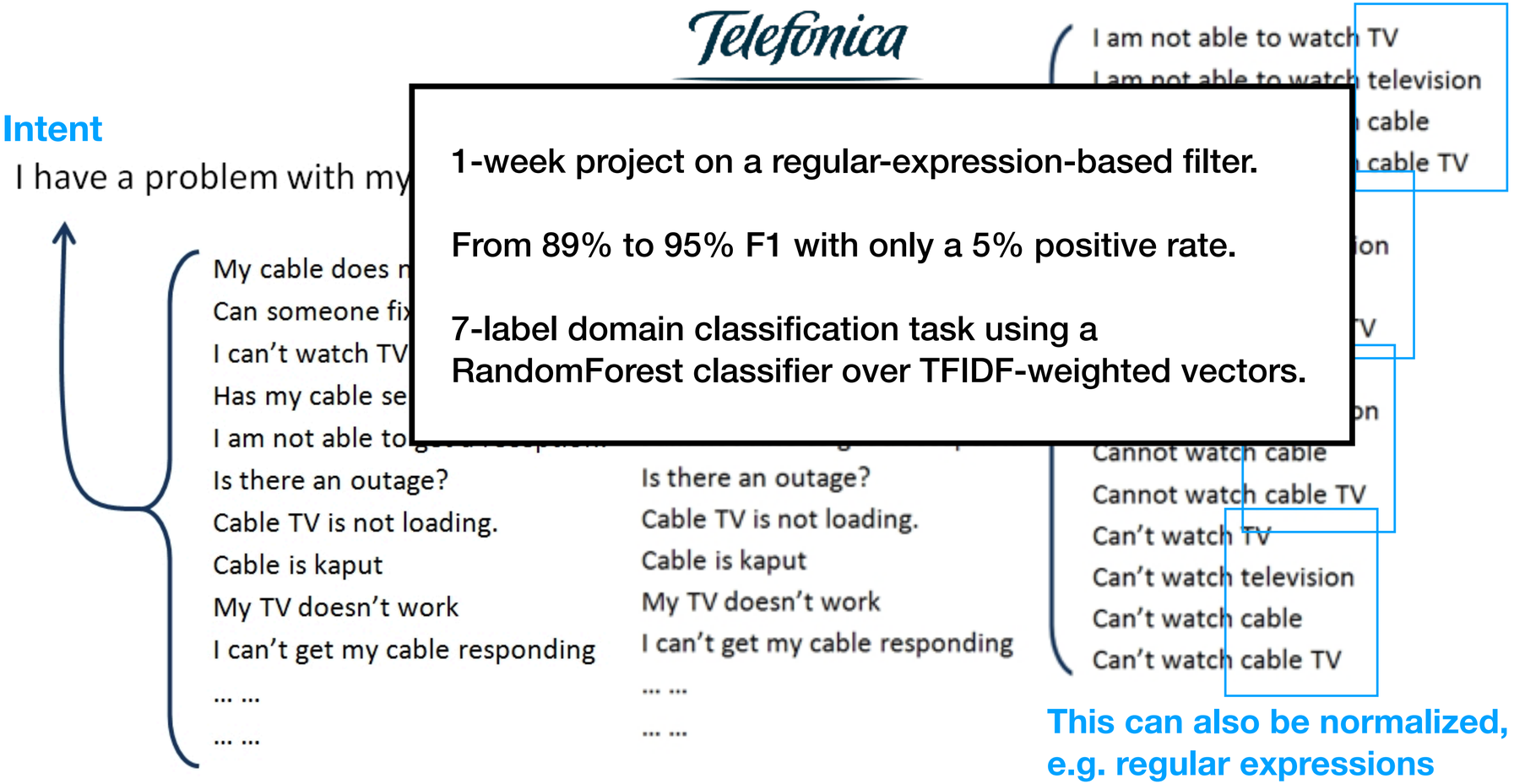
This can also be normalized, e.g. regular expressions

So, let's be real...

lexical variants:
synonyms, paraphrases

Entity

Intent



1-week project on a regular-expression-based filter.

From 89% to 95% F1 with only a 5% positive rate.

7-label domain classification task using a RandomForest classifier over TFIDF-weighted vectors.

This can also be normalized,
e.g. regular expressions

So, let's be real...

lexical variants:
synonyms, paraphrases

Telefónica

Intent

I have a problem with my

- My cable does n
- Can someone fix
- I can't watch TV.
- Has my cable ser
- I am not able to get a reception.
- Is there an outage?
- Cable TV is not loading.
- Cable is kaput
- My TV doesn't work
- I can't get my cable responding
- … …
- … …

I am not able to get a reception.
Is there an outage?
Cable TV is not loading.
Cable is kaput
My TV doesn't work
I can't get my cable responding
… …
… …

Entity

- I am not able to watch TV
- I am not able to watch television
- I am not able to watch cable
- … TV

Cannot watch cable
Cannot watch cable TV
Can't watch TV
Can't watch television
Can't watch cable
Can't watch cable TV

This can also be normalized, e.g. regular expressions

Vs. proposal to handle e.g. negation
as a bi-gram/tri-gram feature engineering problem.

Preferably over a normalized feature space
(*can't watch tv, cannot watch cable* > RB-neg watch TV)

So, let's be real...

lexical variants:
synonyms, paraphrases

Telefonica

Intent

I have a problem with my

My cable does n
Can someone fix
I can't watch TV.
Has my cable ser
I am not able to get a reception.
Is there an outage?
Cable TV is not loading.
Cable is kaput
My TV doesn't work
I can't get my cable responding
… …
… …

I am not able to get a reception.
Is there an outage?
Cable TV is not loading.
Cable is kaput
My TV doesn't work
I can't get my cable responding
… …
… …

Entity

I am not able to watch TV
I am not able to watch television
I am not able to watch cable
TV

Cannot watch cable
Cannot watch cable TV
Can't watch TV
Can't watch television
Can't watch cable
Can't watch cable TV

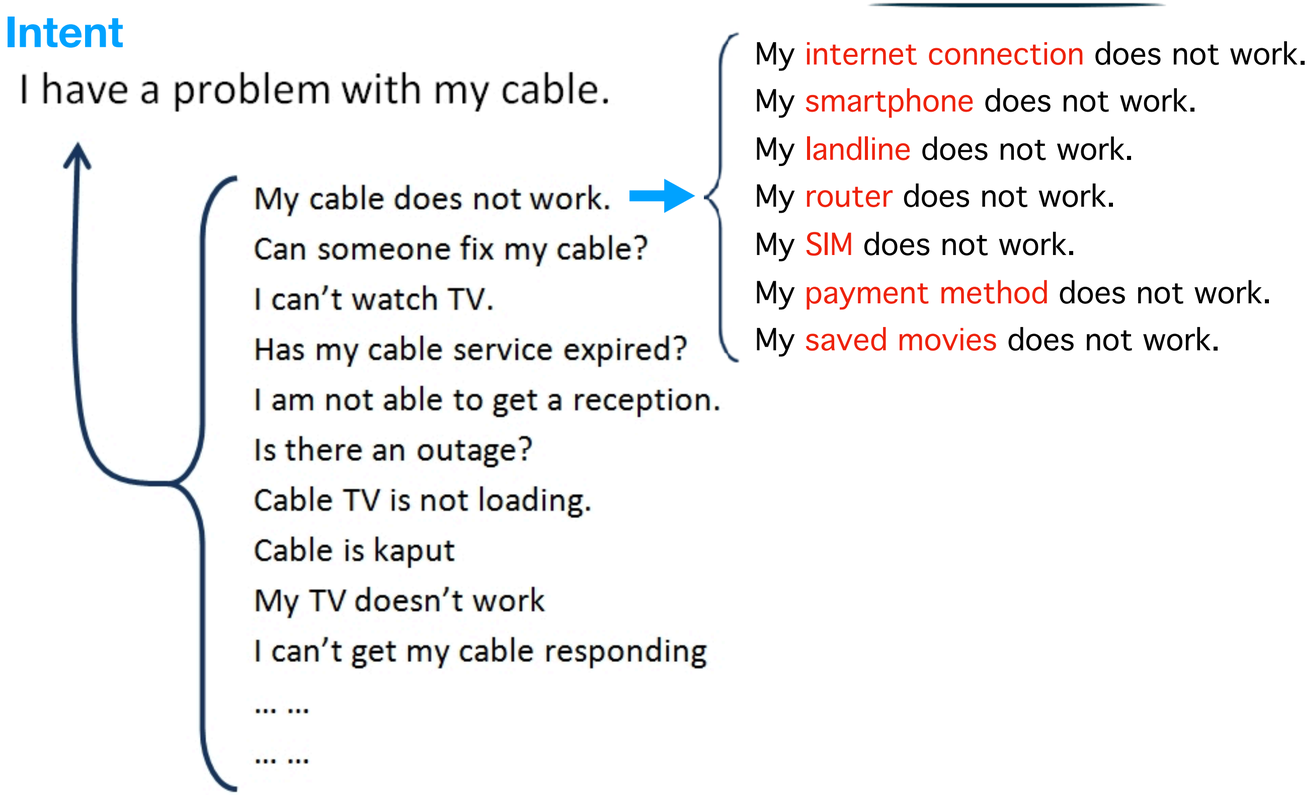
Arguably, providers should be doing it.

All the frameworks currently available on the market force the user to do the work explicitly.

This can also be normalized, e.g. regular expressions

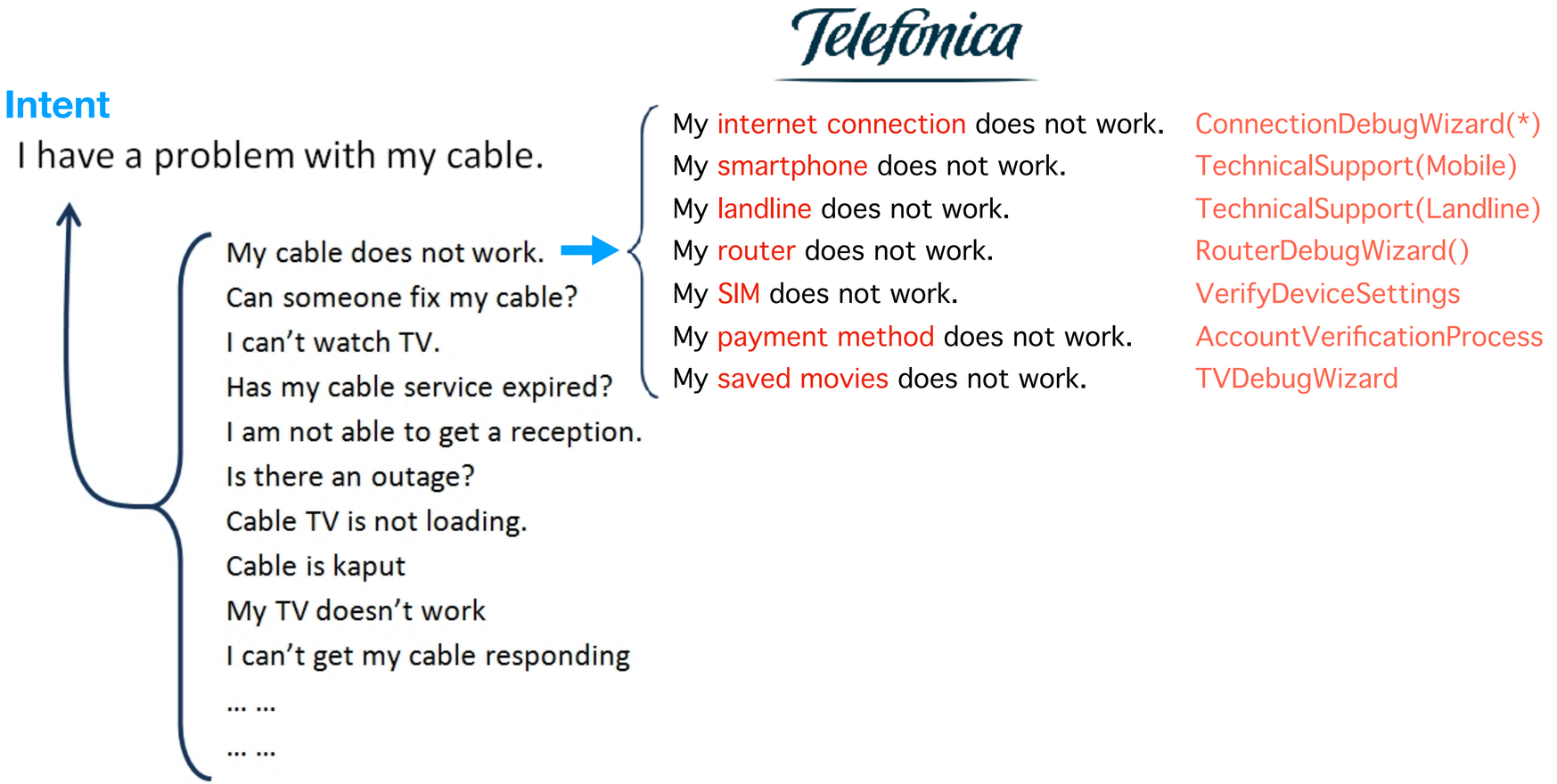
So, let's be real...

lexical variants:
synonyms, paraphrases



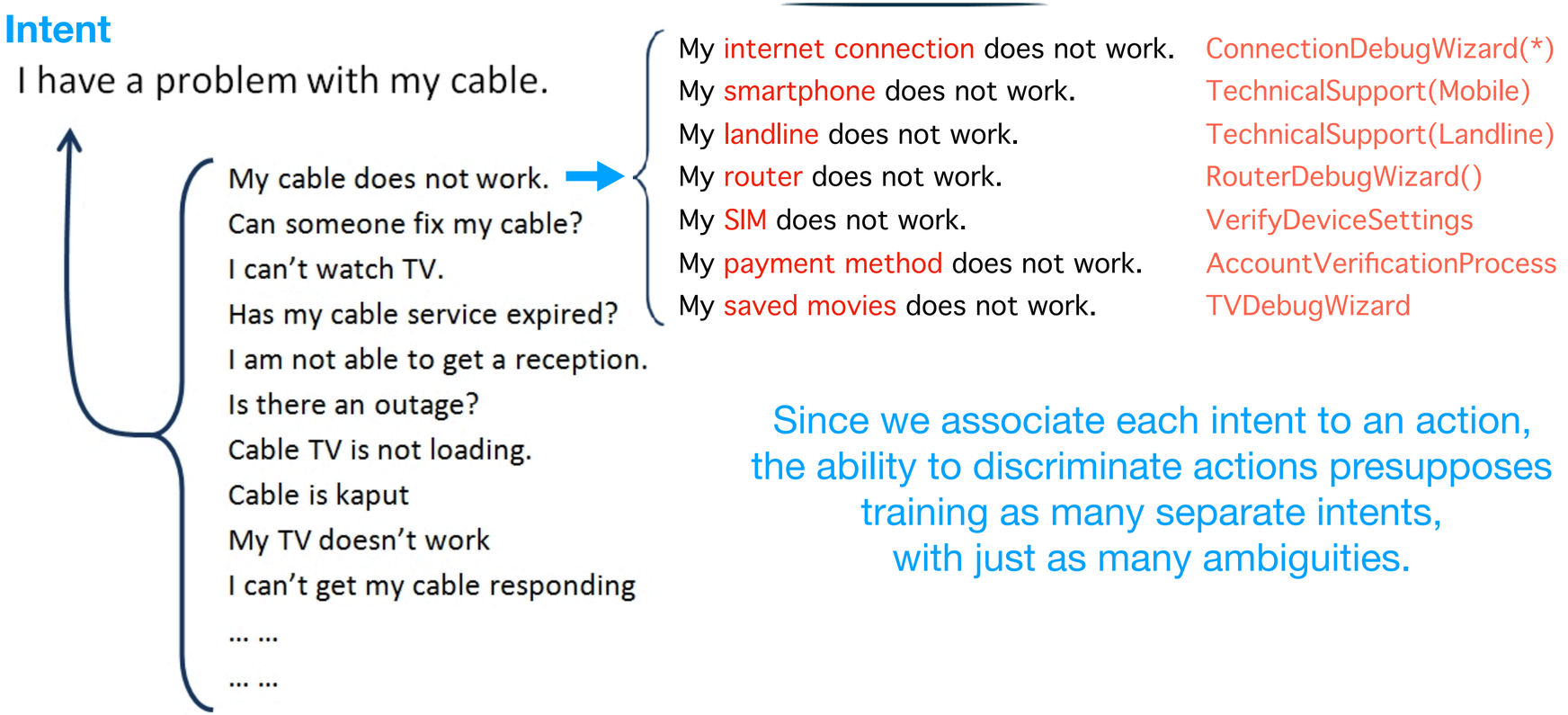
So, let's be real...

lexical variants:
synonyms, paraphrases



So, let's be real...

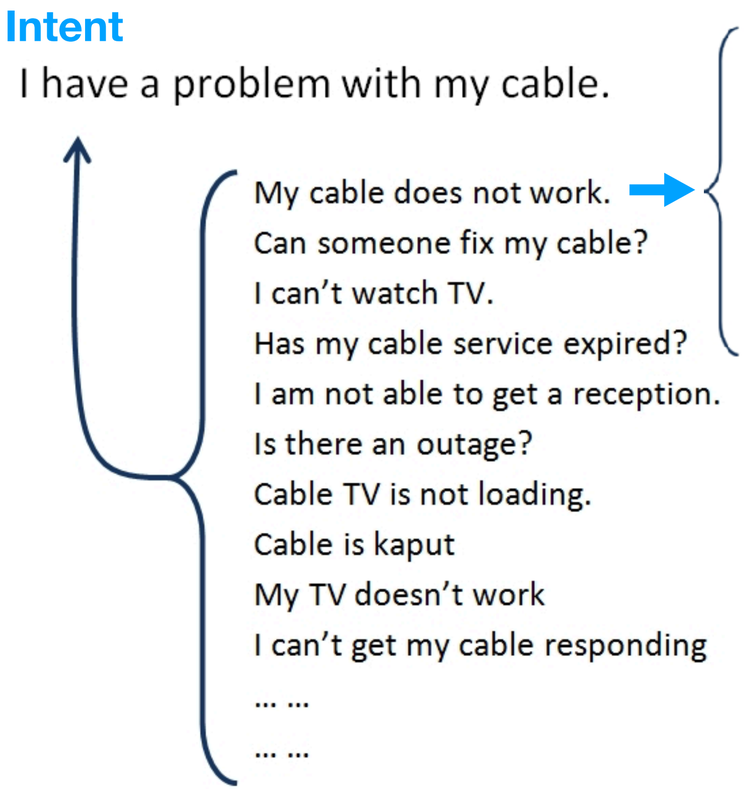
lexical variants:
synonyms, paraphrases



Since we associate each intent to an action,
the ability to discriminate actions presupposes
training as many separate intents,
with just as many ambiguities.

So, let's be real...

lexical variants:
synonyms, paraphrases



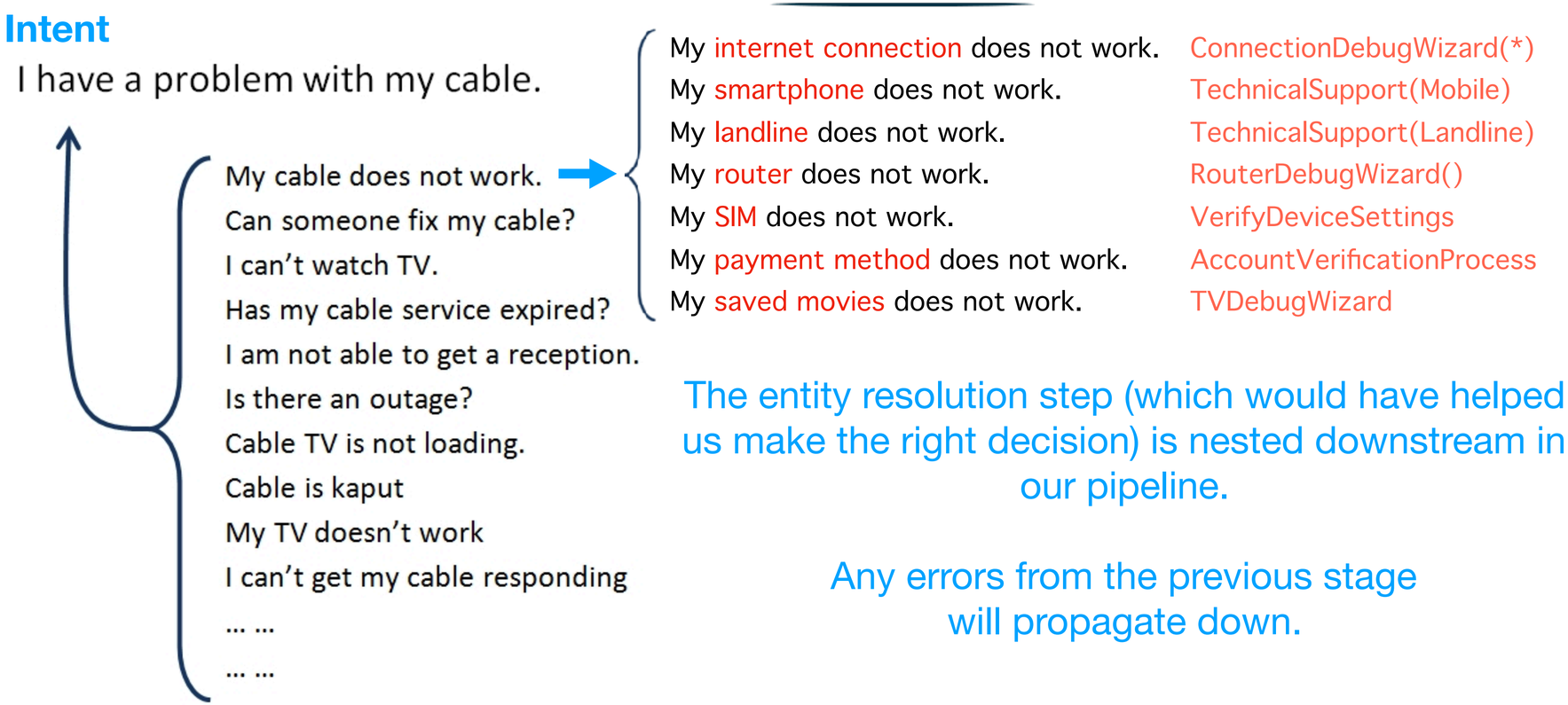
Telefónica

My internet connection does not work.	ConnectionDebugWizard(*)
My smartphone does not work.	TechnicalSupport(Mobile)
My landline does not work.	TechnicalSupport(Landline)
My router does not work.	RouterDebugWizard()
My SIM does not work.	VerifyDeviceSettings
My payment method does not work.	AccountVerificationProcess
My saved movies does not work.	TVDebugWizard

It also presupposes exponentially increasing amounts of training data as we add higher-precision entities to our model (every entity * every intent that applies)

So, let's be real...

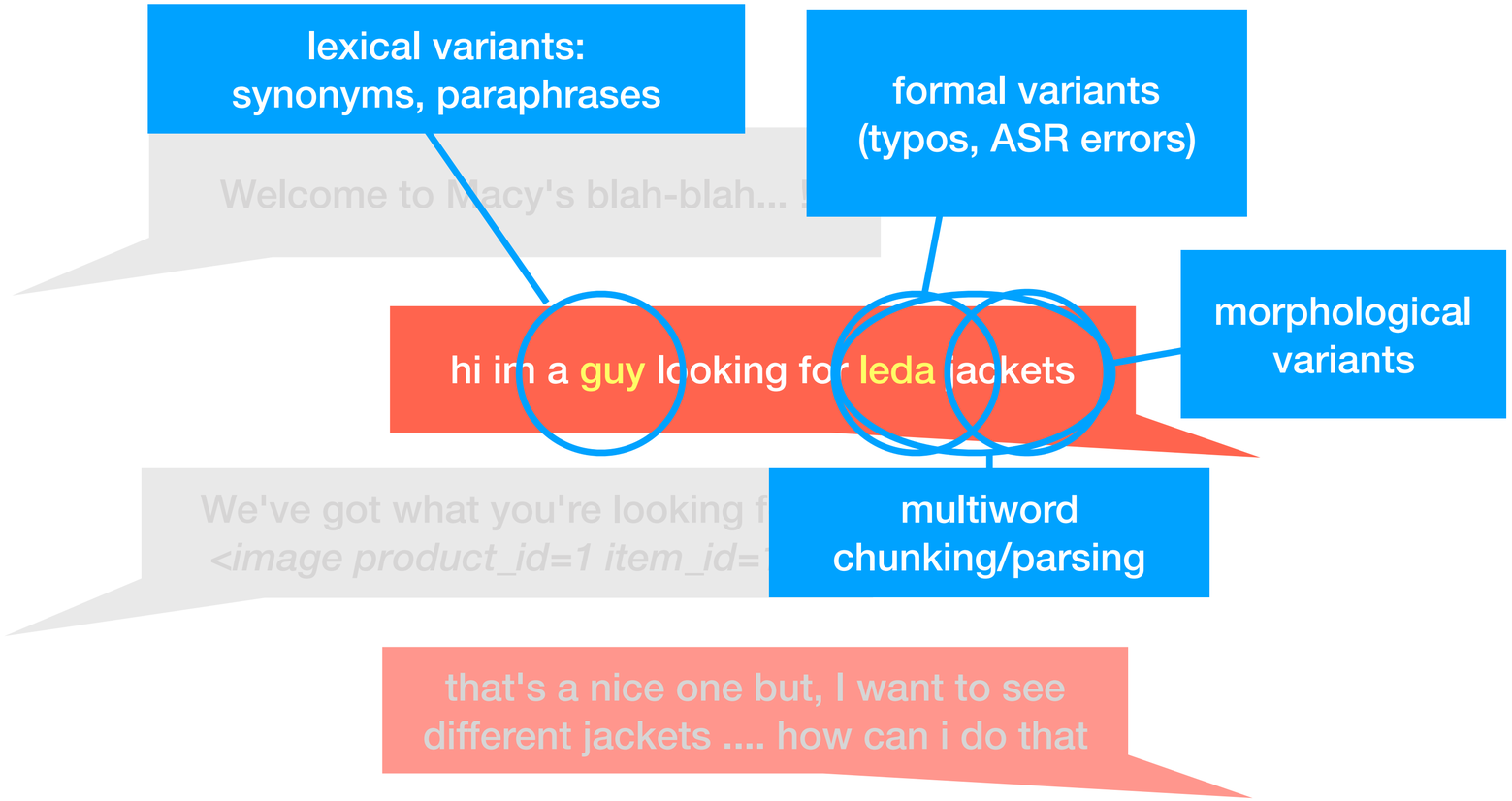
lexical variants:
synonyms, paraphrases



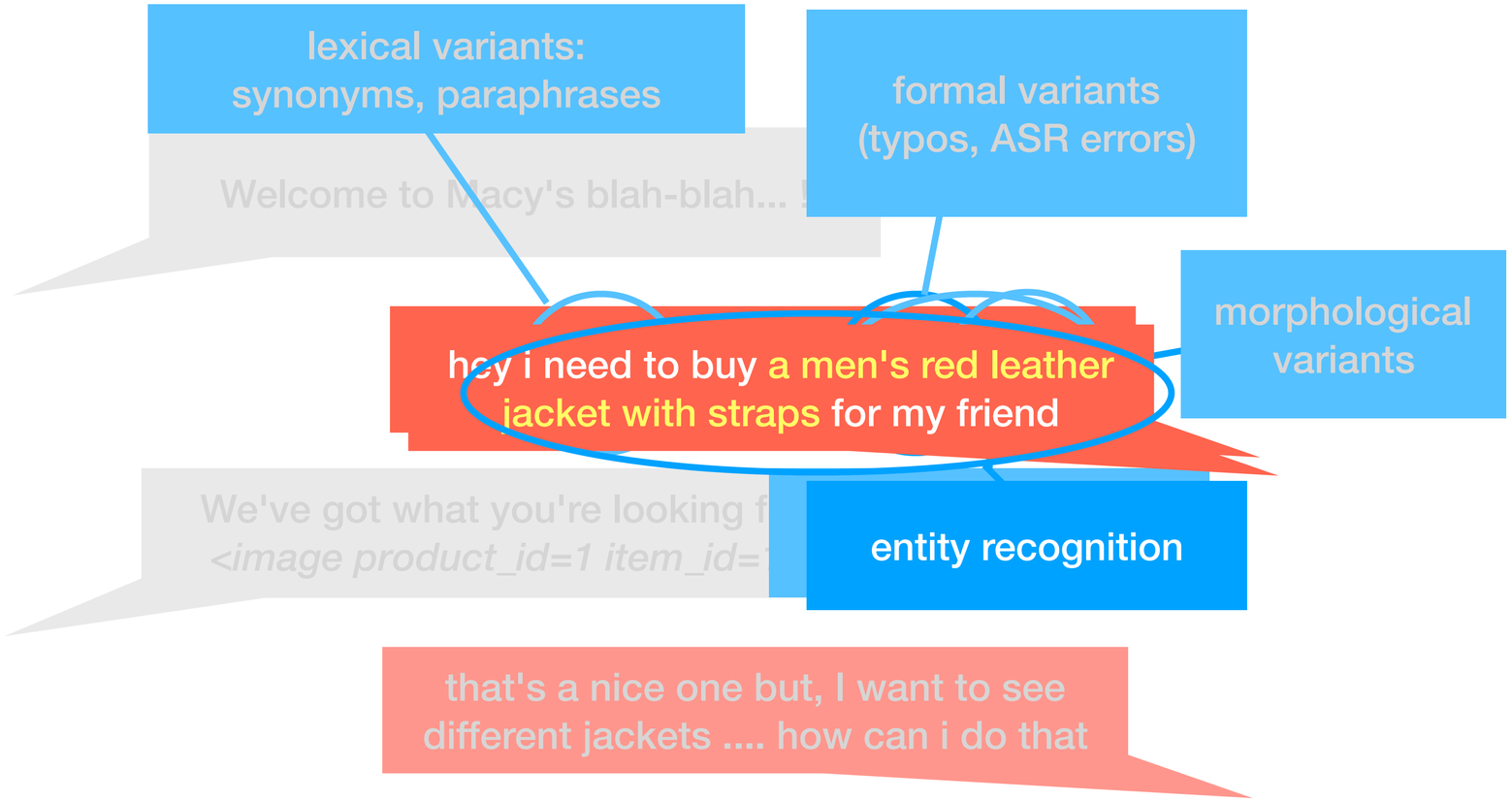
The entity resolution step (which would have helped us make the right decision) is nested downstream in our pipeline.

Any errors from the previous stage will propagate down.

So, let's be real...



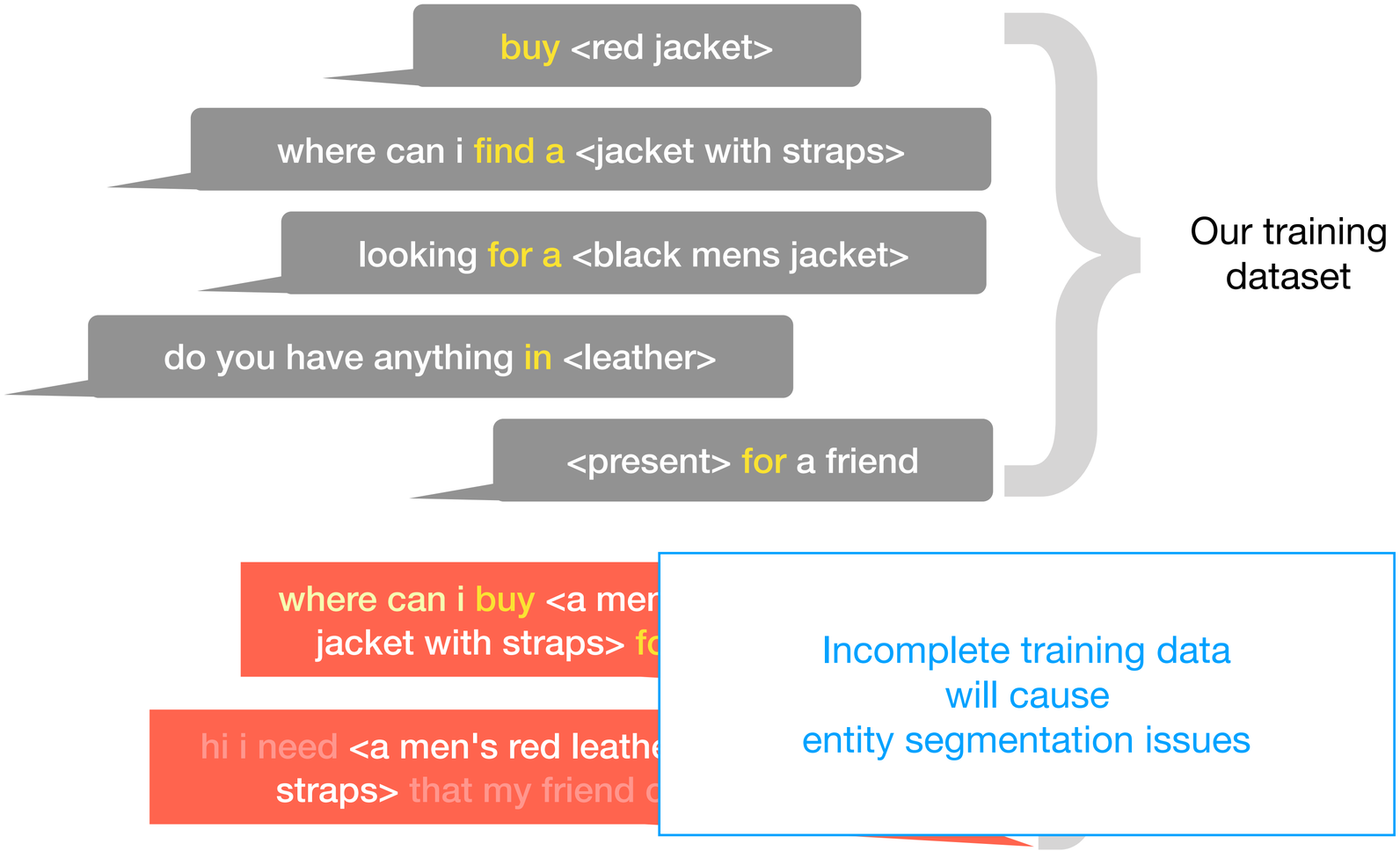
So, let's be real...



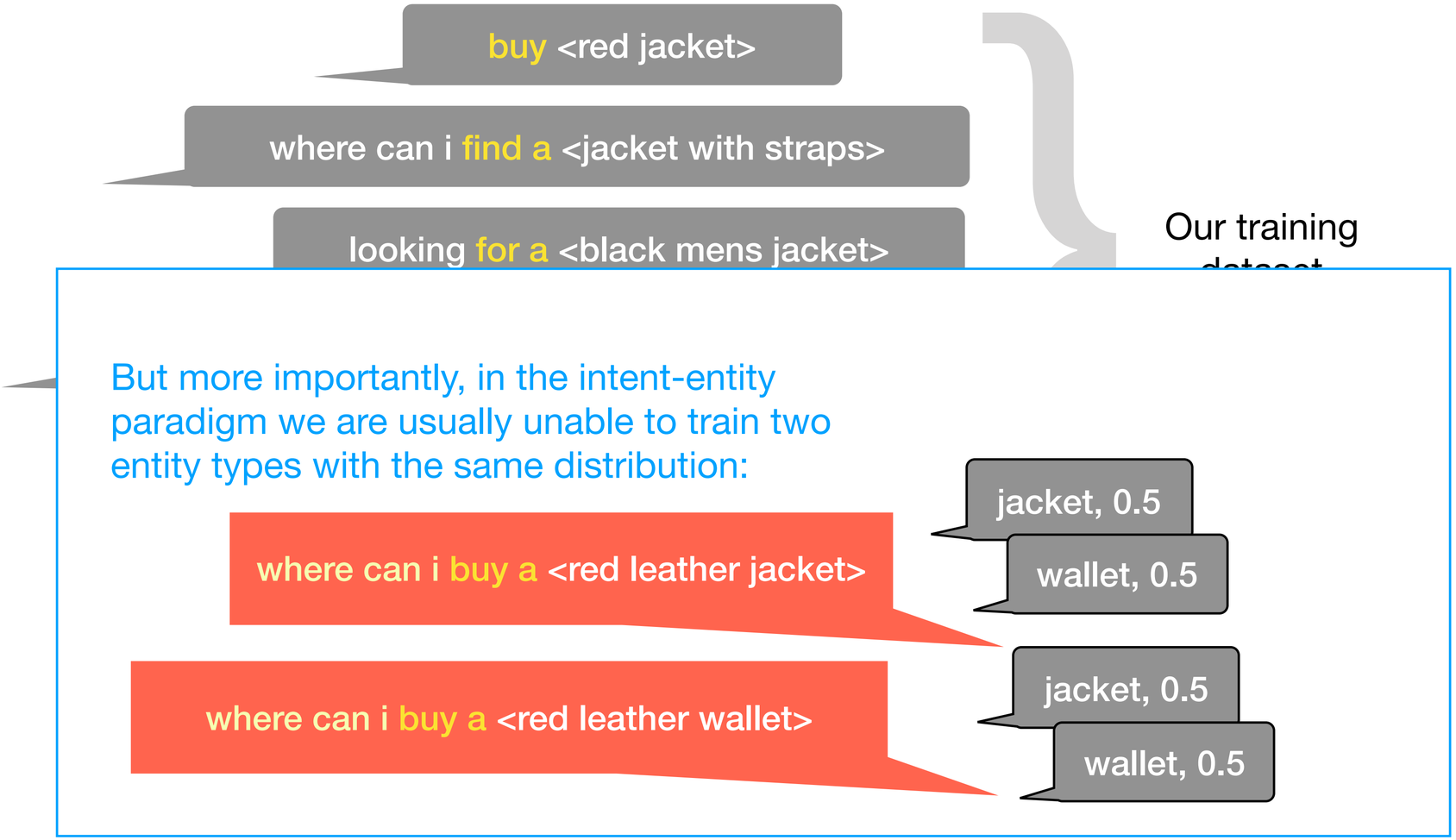
So, let's be real...

So, let's be real...

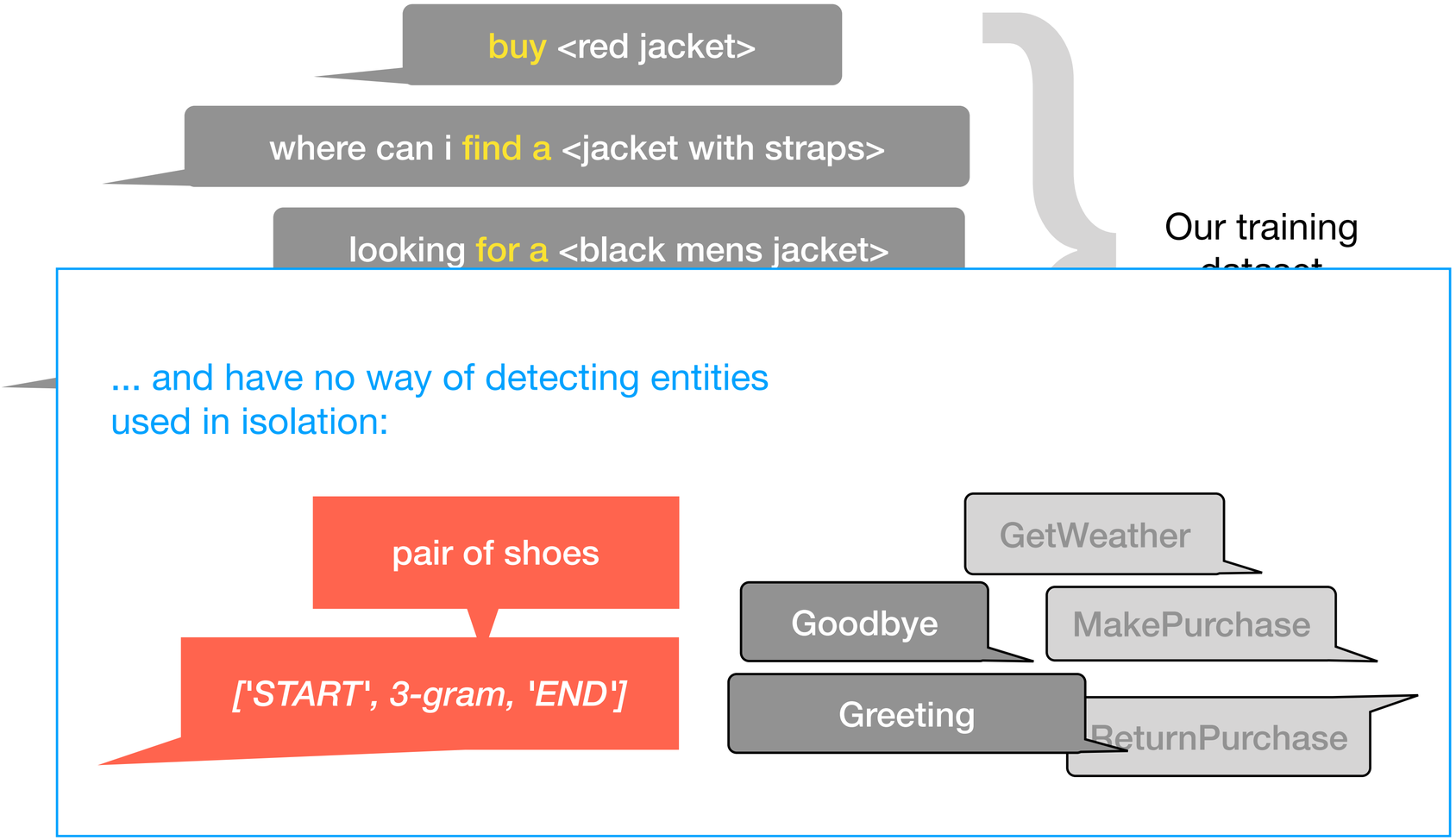
So, let's be real...



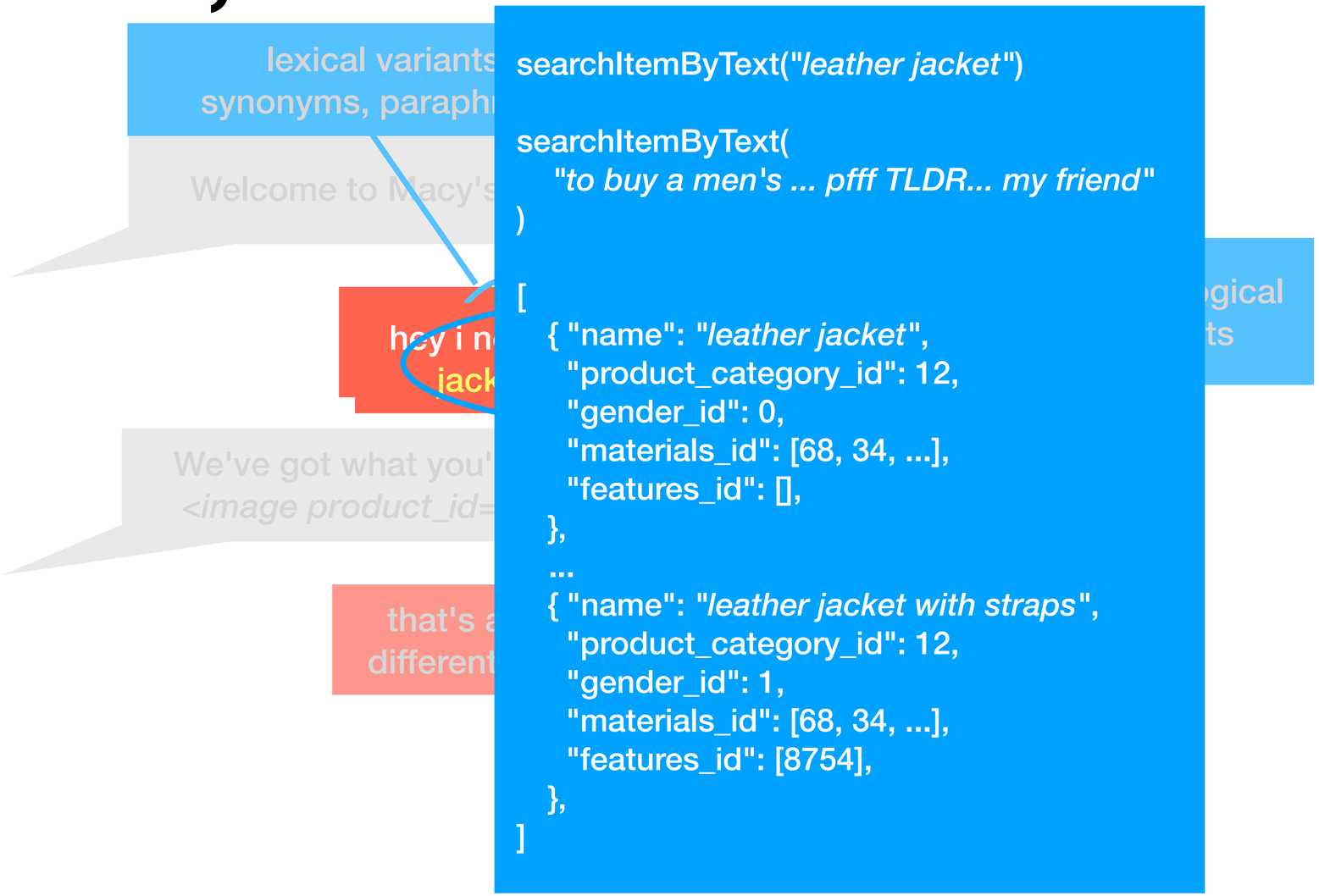
So, let's be real...



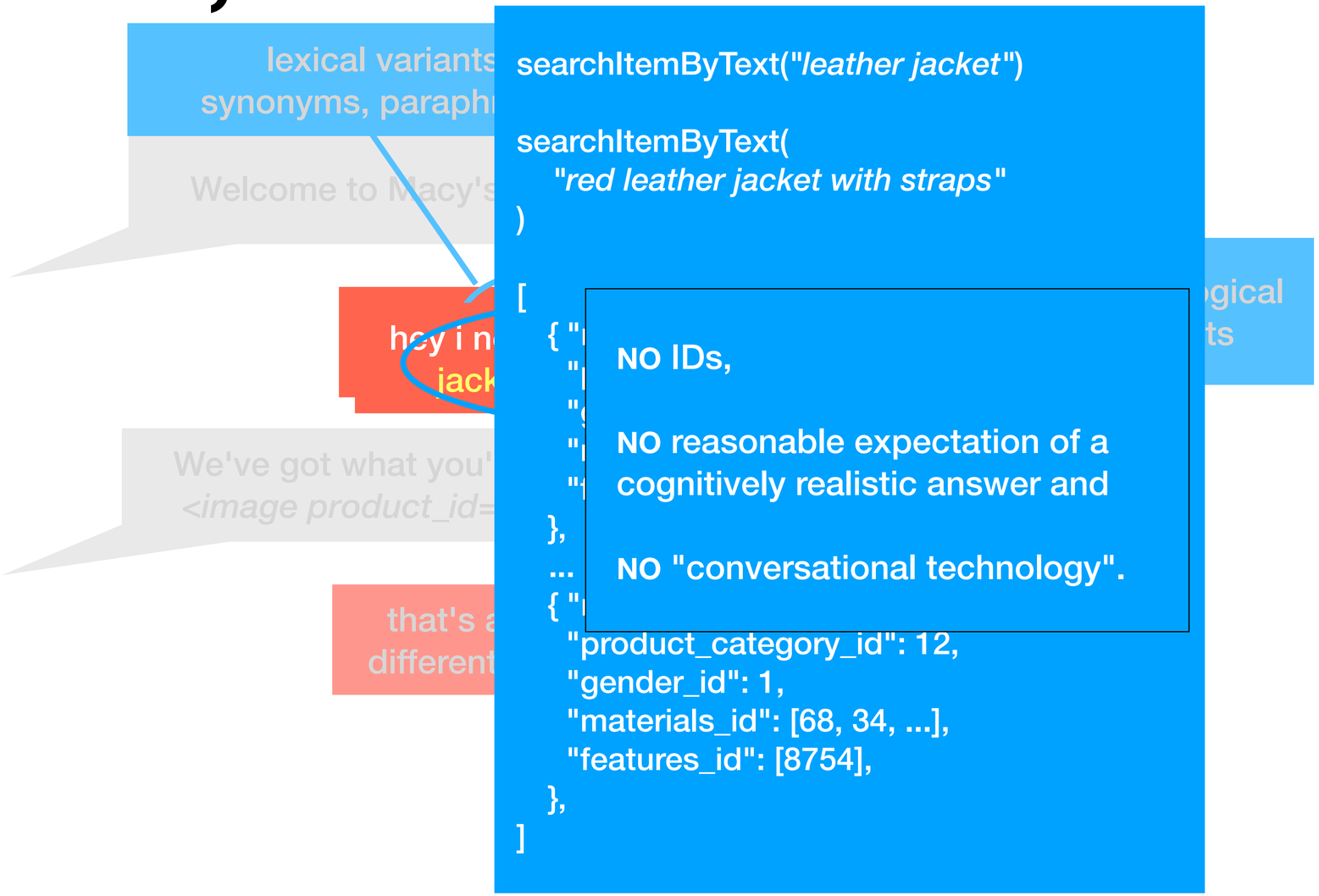
So, let's be real...



So, let's be real...



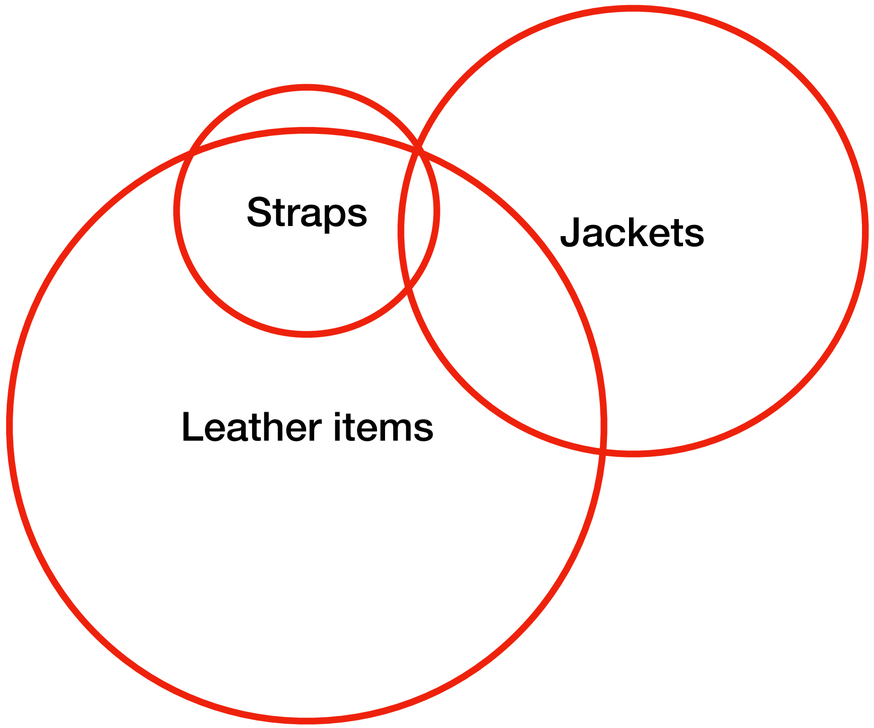
So, let's be real...



So, let's be real...

hey i need to buy a men's red leather jacket with straps for my friend

Since the conditions express an AND logical operator and we will search for an item fulfilling all of them, we could actually search for the terms in the conjunction in any order.



So, let's be real...

hey i need to buy a men's red leather jacket with straps for my friend

Given

- any known word, denoted by w,
- the vocabulary V of all known words, such that $w_i \in V$ for any i,
- search space D, where d denotes every document in a collection D such that $d \in D$ and $d = \{w_i, w_{i+1}, ..., w_n\}$, such that $w_{i \leq x \leq n} \in V$,
- a function $filter(D, w_x)$ that returns a subset of D, Dw_x where $w_x \in d$ is true for every d: $d \in Dw_x$, and
- query q = {"men's", 'red', 'leather', 'jacket', 'straps'},

So, let's be real...

hey i need to buy a men's red leather jacket with straps for my friend

```
do filter(filter(filter(filter(
    filter(D, 'leather'), 'straps'), "men's"), 'jacket'),'red'
)   =

do filter(filter(filter(filter(
    filter(D, 'straps'), 'red'), 'jacket'), 'leather'),"men's"
)   =

do filter(filter(filter(filter(
    filter(D, "men's"), 'leather'), 'red'), 'straps'),'jacket'
)   =
```

So, let's be real...

hey i need to buy a men's red leather jacket with straps for my friend

Given query q = {"men's", 'red', 'leather', 'jacket', 'straps'},
find the x: $x \in q$ that satisfies:

containsSubstring(x, 'leather') ^
containsSubstring(x, 'straps') ^
...
containsSubstring(x, "men's")

We cannot use more complex predicates because, in this scenario, the system has **no** understanding of the internal structure of the phrase, its semantic dependencies, or its syntactic dependencies.

As a result, it will lack the notion of what counts as a partial match.

So, let's be real...

hey i need to buy a men's red leather jacket with straps for my friend

The following partial matches will be virtually indistinguishable:

- leather **straps** for men's *watches* (3/5)
- men's *vintage* red leather ***shoes*** (3/5)
- red ***wallet*** with leather straps (3/5)
- red leather **jackets** with *hood* (3/5)

So, let's be real...

hey i need to buy a men's red leather jacket with straps for my friend

Given

- any known word, denoted by *w*,
- the vocabulary *V* of all kno
- search space *D,* where *d* d ion *D* such that *d* ⊂ *D* and *d* = {*w*
- a function ***filter(D, w_x)*** that $w_x \in d$ is true for every *d*: *d* ∈ D_{w_x}, a
- query *q* = {"men's", 'red', '

This is our problem: we're using a function that treats equally every w_x.

do filter(filter(filter(filter(
    filter(D, "*men's*"), *'red'), 'leather'), 'jacket'),'straps'
)

So, let's be real...

hey i need to buy a men's red leather jacket with straps for my friend

Given

- any known word, denoted b
- the vocabulary *V* of all know
- search space *D,* where *d* de
  such that $d \in D$ and $d = \{w_i$
- a function ***filter(D, w_x)*** that
  true for every *d*: $d \in D_{w_x}$, an
- query *q* = {"men's", 'red', 'l

do filter(filter(filter(filter(
    filter(D, "*men's*"), *'red'), 'le*
)

We need a function with at least one additional parameter, r_x

filter(D, w_x, r_x)

where

r_x denotes the depth at which the dependency tree headed by w_x attaches to the node being currently evaluated.

So, let's be real...

http://nlp.stanford.edu:8080/parser/index.jsp (as of September 19th 2017)

Your query
i need to buy a men's red leather jacket with straps for my friend

Tagging
```
i/FW need/VBP to/TO buy/VB a/DT men/NNS 's/POS red/JJ leather/NN jacket/NN with/IN straps/
NNS for/IN my/PRP$ friend/NN
```

Parse
```
(ROOT
  (S
    (NP (FW i))
    (VP (VBP need)
      (S
        (VP (TO to)
          (VP (VB buy)
            (NP
              (NP (DT a) (NNS men) (POS 's))
              (JJ red) (NN leather) (NN jacket))
            (PP (IN with)
              (NP
                (NP (NNS straps))
                (PP (IN for)
                  (NP (PRP$ my) (NN friend)))))))))))
```

So, let's be real...

http://nlp.stanford.edu:8080/parser/index.jsp (as of September 19th 2017)

Your query
i need to buy a men's red leather jacket with straps for my friend

Tagging
```
i/FW need/VBP to/TO buy/VB a/DT men/NNS 's/POS red/JJ leather/NN jacket/NN with/IN straps/
NNS for/IN my/PRP$ friend/NN
```

Parse
```
(ROOT
  (S
    (NP (FW i))
    (VP (VBP need)
      (S
        (VP (TO to)
          (VP (VB buy)
            (NP
              (NP (DT a) ((NNS men) (POS 's))
              ((JJ red) (NN leather)) (NN jacket)))
            (PP (IN with)
              (NP
                (NP (NNS straps))
            (PP (IN for)
              (NP (PRP$ my) (NN friend)))))))))))
```

So, let's be real...

http://nlp.stanford.edu:8080/parser/index.jsp **(as of September 19th 2017)**

Your query
i need to buy a men's red leather jacket with straps for my friend

Tagging
```
i/FW need/VBP to/TO buy/VB a/DT men/NNS 's/POS red/JJ leather/NN jacket/NN with/IN straps/
NNS for/IN my/PRP$ friend/NN
```

Parse
```
(ROOT
  (S
    (NP (FW i))
    (VP (VBP need)
      (S
        (VP (TO to)
          (VP (VB buy)
            (NP
              (NP (DT a) ((NNS men) (POS 's))
              ((JJ red) (NN leather)) (NN jacket)))
            (PP (IN with)
              (NP
                (NP (NNS straps))
            (PP (IN for)
              (NP (PRP$ my) (NN friend)))))))))))
```

(red, leather)

(leather, jacket)

(jacket, **buy**)

So, let's be real...

hey i need to buy a men's red leather jacket with straps for my friend

Given

- any known word, denoted by *w*
- the vocabulary *V* of all known w
- search space *D,* where *d* denot
  such that $d \subset D$ and $d = \{w_i, w_i$
- a function ***filter(D, w_x)*** that retu
  true for every *d:* $d \in D_{w_x}$, and
- query *q* = {"men's", 'red', 'leath

do filter(filter(filter(filter(
    filter(D, "*men's*"), *'red'), 'leathe*
)

The output of this function will no longer be a subset of search results but a ranked list.

This list can be naively ranked by

$$\sum_x^{|t|} (1 / r_x)$$

although much more advanced scoring functions can be used.

So, let's be real...

hey i need to buy a men's red leather jacket with straps for my friend

Given:

q: "*red leather jacket*"

a_1: "*black leather jacket*"

a_2: "*red leather wallet*"

So, let's be real...

hey i need to buy a men's red leather jacket with straps for my friend

Given:

q: "*red leather jacket*"

a_1: "*black leather jacket*"

a_2: "*red leather wallet*"

```
>>> t1 = dependencyTree(a1)
>>> t1.depth("black")
3
>>> t1.depth("jacket")
1
```

So, let's be real...

hey i need to buy a men's red leather jacket with straps for my friend

Given:
q: "*red leather jacket*"
a_1: "*black leather jacket*"
a_2: "*red leather wallet*"

>>> results = filter(D, "*jacket*", t1.depth("*jacket*"))
>>> results
[($Jacket_1$, 1.0),
($Jacket_2$, 1.0),
...,
($Jacket_n$, 1.0)
]

So, let's be real...

hey i need to buy a men's red leather jacket with straps for my friend

Given:

q: "*red leather jacket*"

a_1: "*black leather jacket*"

a_2: "*red leather wallet*"

```
>>> results = filter(results, "leather", t1.depth("leather"))
>>> results
[  ( Jacket_1, 1.5 ),
   ( Jacket_2, 1.5 ),
  ...,
   ( Jacket_n - x, 1.5 )
]
```

So, let's be real...

hey i need to buy a men's red leather jacket with straps for my friend

Given:
q: "*red leather jacket*"
a_1: "*black leather jacket*"
a_2: "*red leather wallet*"

>>> results = filter(results, "*red*", t1.depth("*red*"))
>>> results
[($Jacket_1$, 1.5),
($Jacket_2$, 1.5),
...,
($Jacket_{n-x}$, 1.5)
]

So, let's be real...

hey i need to buy a men's red leather jacket with straps for my friend

Given:

q: "*red leather jacket*"

a_1: "*black leather jacket*"

a_2: "*red leather wallet*"

```
>>> t2 = dependencyTree(a2)
>>> t2.depth("red")
3
>>> t2.depth("jacket")
None
```

So, let's be real...

hey i need to buy a men's red leather jacket with straps for my friend

Given:
  q: "*red leather jacket*"
  a_1: "*black leather jacket*"
  a_2: "*red leather wallet*"

  >>> results = filter(D, "*jacket*", t2.depth("*jacket*"))
  >>> results
  []

So, let's be real...

hey i need to buy a men's red leather jacket with straps for my friend

Given:
q: "*red leather jacket*"
a_1: "*black leather jacket*"
a_2: "*red leather wallet*"

>>> results = filter(results, "*leather*", t2.depth("*leather*"))
>>> results
[($Wallet_1$, 0.5),
($Wallet_2$, 0.5),
...,
($Strap_n$, 0.5)
]

So, let's be real...

hey i need to buy a men's red leather jacket with straps for my friend

Given:

q: "*red leather jacket*"

a_1: "*black leather jacket*"

a_2: "*red leather wallet*"

```
>>> results = filter(results, "red", t2.depth("red"))
>>> results
[  ( Wallet1, 0.83 ),
   ( Wallet2, 0.83 ),
  ...,
   ( Strapn - x, 0.83 )
]
```

So, let's be real...

hey i need to buy a men's red leather jacket with straps for my friend

Results for q_1:
"*black leather jacket*"

[($Jacket_1$, 1.5),
($Jacket_2$, 1.5),
...,
($Jacket_{n-x}$, 1.5)
]

Results for q_2:
"*red leather wallet*"

[($Wallet_1$, 0.83),
($Wallet_2$, 0.83),
...,
($Strap_{n-x}$, 0.83)
]

So, let's be real...

hey i need to buy a men's red leather jacket with straps for my friend

However, the dependency tree is not always available, and it often contains parsing issues (cf. Stanford parser output).

An evaluation of parser robustness over noisy input reports performances down to an average of 80% across datasets and parsers.

Hashemi, Homa B., and Rebecca Hwa. "An Evaluation of Parser Robustness for Ungrammatical Sentences." *EMNLP*. 2016.

So, let's be real...

hey i need to buy a men's red leather jacket with straps for my friend

In order to make this decision

 [[red leather] jacket]

So, let's be real...

hey i need to buy a men's red leather jacket with straps for my friend

In order to make this decision

[[red leather] jacket]

[red [leather jacket]]

So, let's be real...

hey i need to buy a men's red leather jacket with straps for my friend

In order to make this decision

[[red leather] jacket]

[red [leather jacket]]

[men's [leather jacket]]

So, let's be real...

hey i need to buy a men's red leather jacket with straps for my friend

In order to make this decision

[[red leather] jacket]

[red [leather jacket]]

[men's [leather jacket]]

😱 [[men's leather] jacket]

So, let's be real...

hey i need to buy a men's red leather jacket with straps for my friend

In order to make this decision

[[red leather] jacket]

[red [leather jacket]]

[men's [leather jacket]]

😱 [[men's leather] jacket]

... the parser already needs as much information as we can provide regarding head-modifier attachments.

So, let's be real...

hey i need to buy a men's red leather jacket with straps for my friend

In order to make this decision

[[red leather] jacket]	2 2 1
[red [leather jacket]]	2 1 1
[men's [leather jacket]]	2 1 1
[[men's leather] jacket]	2 2 1

So, let's be real...

hey i need to buy a men's red leather jacket with straps for my friend

In order to make this decision

[[red leather] jacket]	2 2 1
[red [leather jacket]]	2 1 1
[men's [leather jacket]]	2 1 1
[[men's leather] jacket]	2 2 1

Same pattern, different score. How to assign a probability such that less plausible results are buried at the bottom of the hypotheses space? (not only for display purposes, but for dynamic programming reasons)

So, let's be real...

hey i need to buy a men's red leather jacket with straps for my friend

In order to make this decision

[[red leather] jacket]	2 2 1	p(*red* | *jacket*) ~
[red [leather jacket]]	2 1 1	p(*red* | *leather*)
[men's [leather jacket]]	2 1 1	p(*men's* | *jacket*) >
[[men's leather] jacket]	2 2 1	p(*men's* | *leather*)

So, let's be real...

hey i need to buy a men's red leather jacket with straps for my friend

That's a lot of data to model. Where will we get it from?

[[red leather] jacket]	2 2 1
[red [leather jacket]]	2 1 1
[men's [leather jacket]]	2 1 1
[[men's leather] jacket]	2 2 1

So, let's be real...

hey i need to buy a men's red leather jacket with straps for my friend

That's a lot of data to model. Where will we get it from?

[[red leather] jacket]	2 2 1	p(*Color* | *ClothingItem*) ~
[red [leather jacket]]	2 1 1	p(*Color* | *Material*)
[men's [leather jacket]]	2 1 1	p(*Person* | *ClothingItem*) >
[[men's leather] jacket]	2 2 1	p(*Person* | *Material*)

So, let's be real...

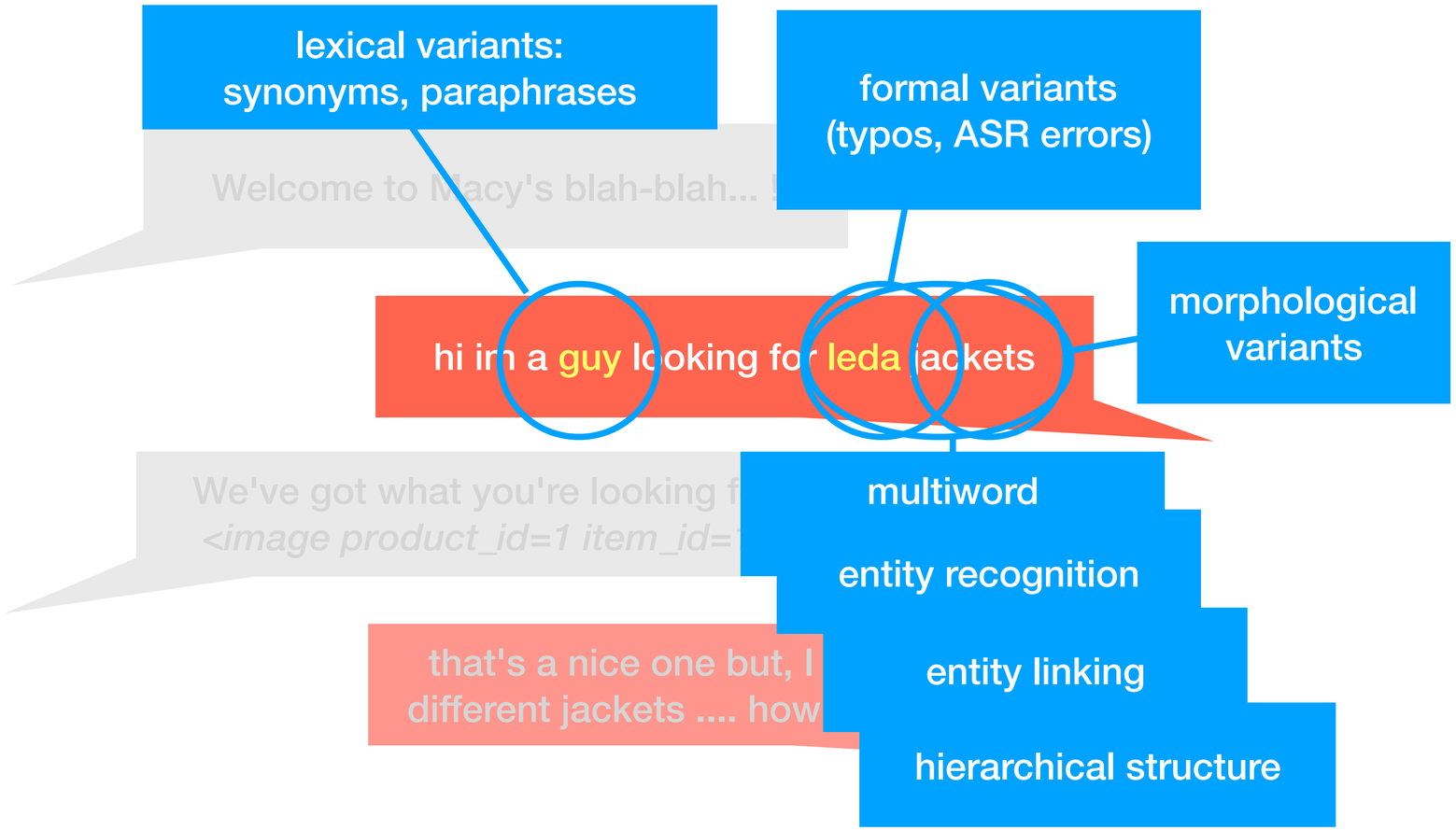
hey i need to buy a men's red leather jacket with straps for my friend

If our taxonomy is well built, semantic associations acquired for a significant number of members of each class will propagate to new, unseen members of that class.

With a sufficiently rich NER detection pipeline and a semantic relation database large enough, we can rely on simple inference for a vast majority of the disambiguations.

More importantly, we can weight candidate hypotheses dynamically in order to make semantic parsing tractable over many possible syntactic trees.

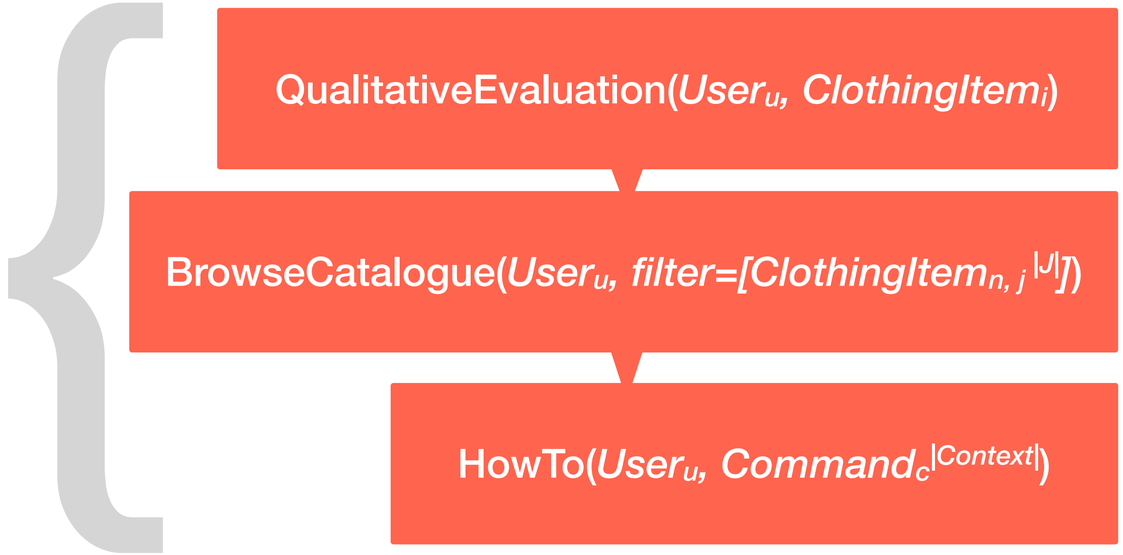
So, let's be real...



So, let's be real...

that's a nice one but, i want to see different jackets .... how can i do that

A single utterance



So, let's be real...

that's a nice one but, i want to see different jackets .... how can i do that

A single utterance

QualitativeEvaluation($User_u$, $ClothingItem_i$)

BrowseCatalogue($User_u$, $filter=[ClothingItem_{n,j}{}^{|J|}]$)

HowTo($User_u$, $Command_c{}^{|Context|}$)

So, let's be real...

that's a nice one but, i want to see different jackets .... how can i do that

$I = [\, i_1, i_2, i_3 \,]$
$L = [\, |\, i_1 \,|, \ |\, i_2 \,|, \ |\, i_3 \,|\,]$
$P = [\, p(\, i_1 \,), p(\, i_2 \,), p(\, i_3 \,)]$

random.choice(I)

x if $x \in I$ ^
length(x) = argmax(L)

x if $x \in I$ ^
p(x) = argmax(P)

QualitativeEvaluation($User_u$, $ClothingItem_i$)

BrowseCatalogue($User_u$, $filter=[ClothingItem_{n,\,j}^{|J|}]$)

HowTo($User_u$, $Command_c^{|Context|}$)

So, let's be real...

that's a nice one but, i want to see different jackets .... how can i do that

I = [i_1, i_2, i_3]
L = [| i_1 |, | i_2 |, | i_3 |]
P = [p(i_1 | q), p(i_2 | q), p(i_3 | q)]

random.choice(I)

x if $x \in I$ ^
length(x) = argmax(L)

x if $x \in I$ ^
p(x | q) = argmax(P)

However, this approach will still suffer from segmentation issues due to passing many more arguments than expected for resolving any specific intent.

So, let's be real...

that's a nice one but, i want to see different jackets .... how can i do that

A standard API will normally provide these definitions as the declaration of its methods.

$$\text{SetPreference}(User_u, \text{SetRating}(ClothingItem_{i,k}, 1))$$

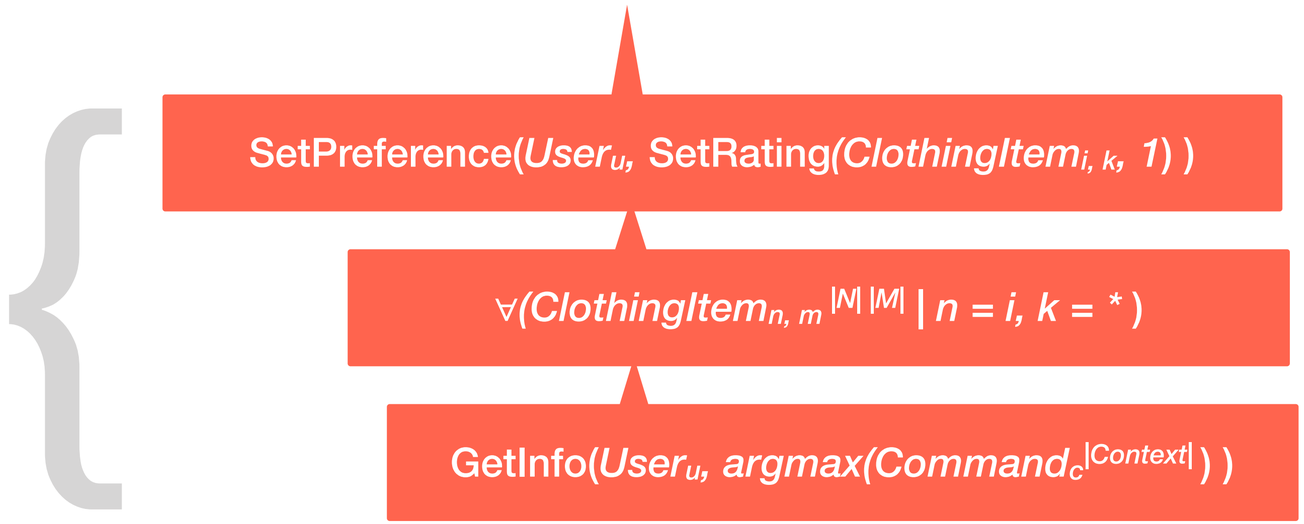
$$\forall(ClothingItem_{n,m}^{|N|\,|M|} \mid n = i, k = *)$$

$$\text{GetInfo}(User_u, argmax(Command_c^{|Context|}))$$

So, let's be real...

that's a nice one but, i want to see different jackets .... how can i do that

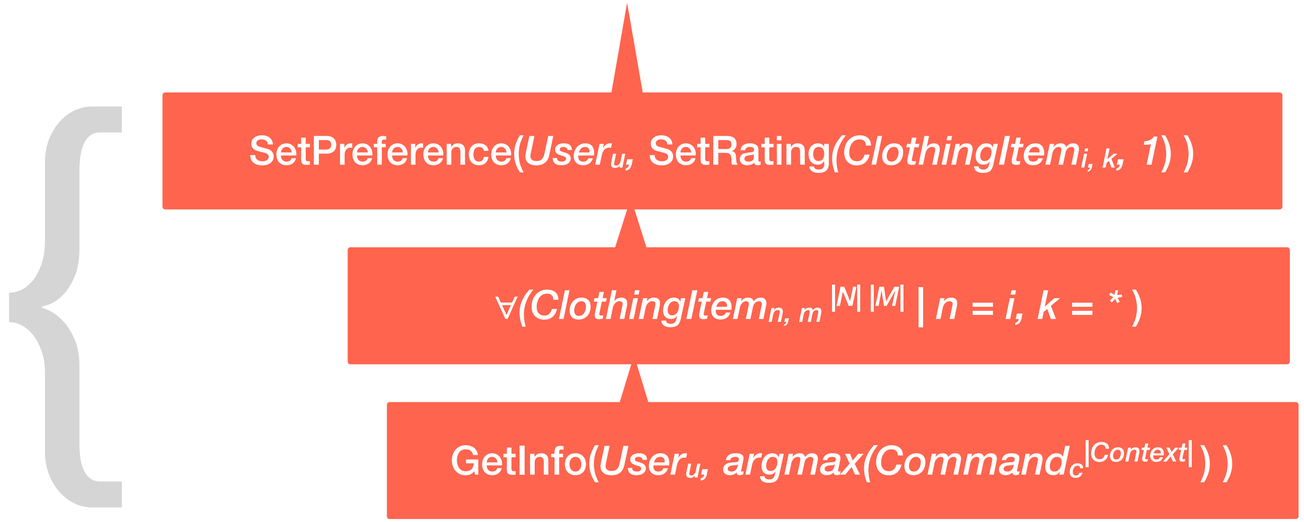
We can map the argument signatures to the nodes of the taxonomy we are using for linguistic inference.



So, let's be real...

that's a nice one but, i want to see different jackets .... how can i do that

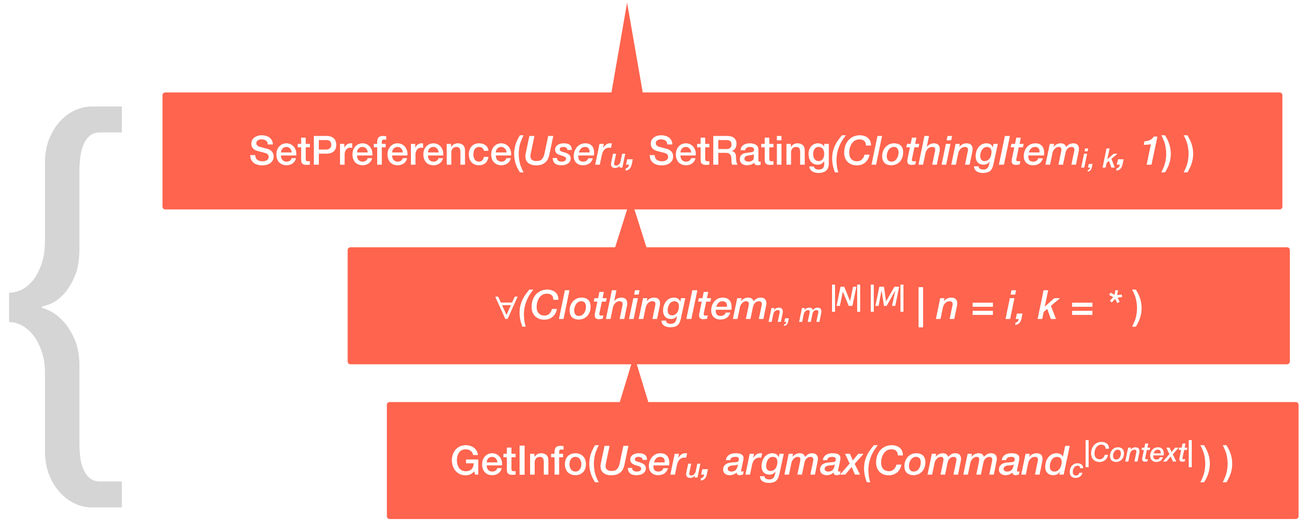
The system will then be able to link API calls to entity parses dynamically.



So, let's be real...

that's a nice one but, i want to see different jackets .... how can i do that

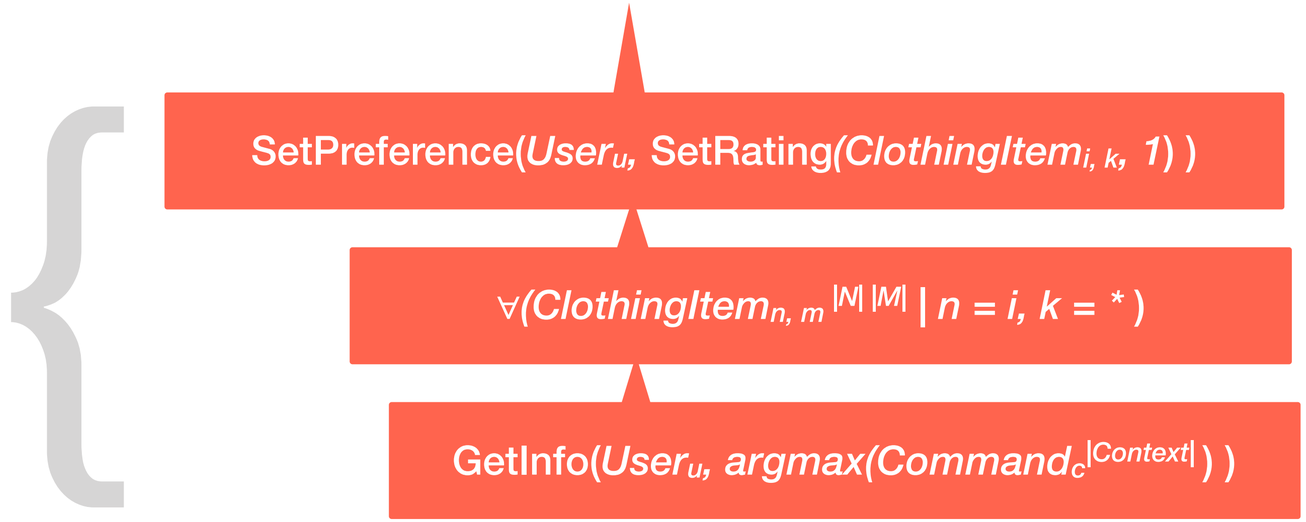
Over time, new methods added to the API will be automatically supported by the pre-existing linguistic engine.



So, let's be real...

that's a nice one but, i want to see different jackets .... how can i do that

Over time, new methods added to the API will be automatically supported by the pre-existing linguistic engine.



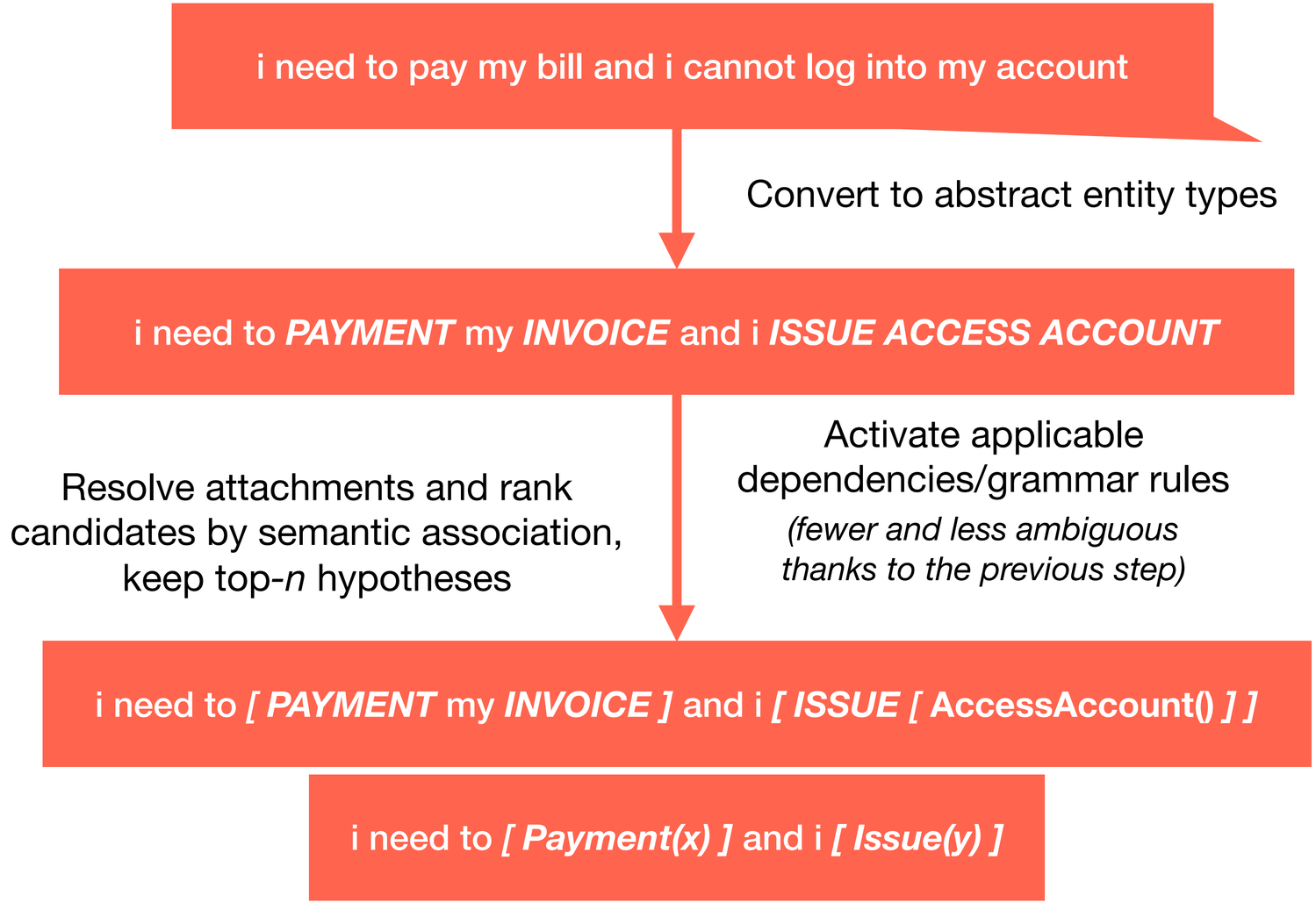
Telefónica 😁

So, let's be real...

If the system can reasonably solve
nested semantic dependencies hierarchically,

it can also be extended to handle multi-intent
utterances in a natural way.

So, let's be real...



So, let's be real...

i need to *[**Payment(x)**]* and i *[**Issue(y)**]*

AND usually joins
ontologically coordinate elements,
e.g. *books **and** magazines* **versus**
**humans and Donald Trump*

So, let's be real...

i need to *[Payment(x)]* and i *[Issue(y)]*

Even if

$$p_0 = p(\,Payment \mid Issue\,) \;>\; u$$

we will still split the intents because here we have

$$p_y = p(\,Payment,\, x \mid Issue,\, y\,)$$

and normally $p_y < p_0$ for any irrelevant case (i.e., the posterior probability of the candidate attachment will not beat the baseline of its own prior probability)

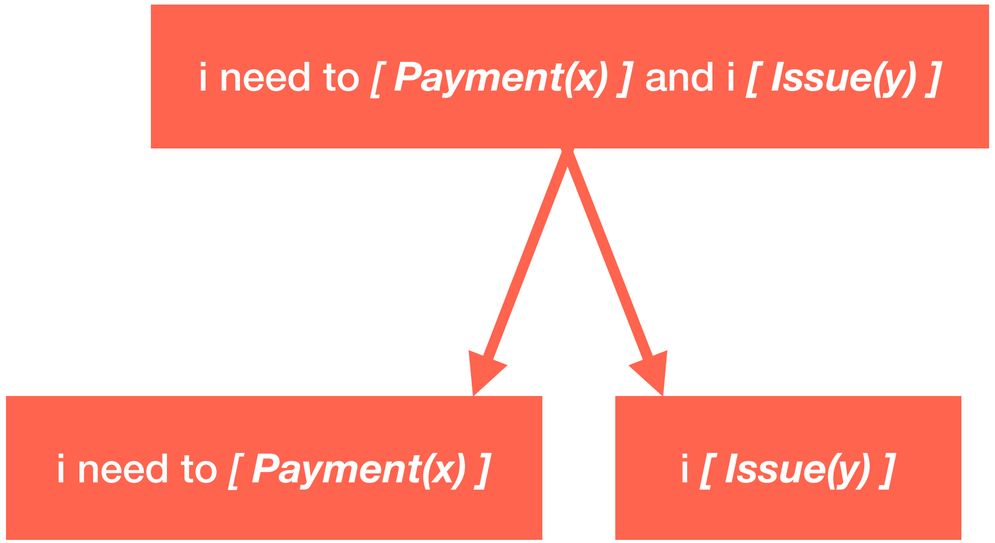
So, let's be real...

i need to *[Payment(x)]* and i *[Issue(y)]*

I.e., *Payment* may have been attached to *Issue* if no other element had been attached to it before.

Resolution order, as given by attachment probabilities, matters.

So, let's be real...



Multi-intents solved 😎

So, let's be real...

no! forget the last thing i said

go back!

third option in the menu

i want to modify the address I gave you

So, let's be real...

We need:

- semantic association measures between the leaves of the tree in order to attach them to the right nodes and be able to stop growing a tree at the right level,

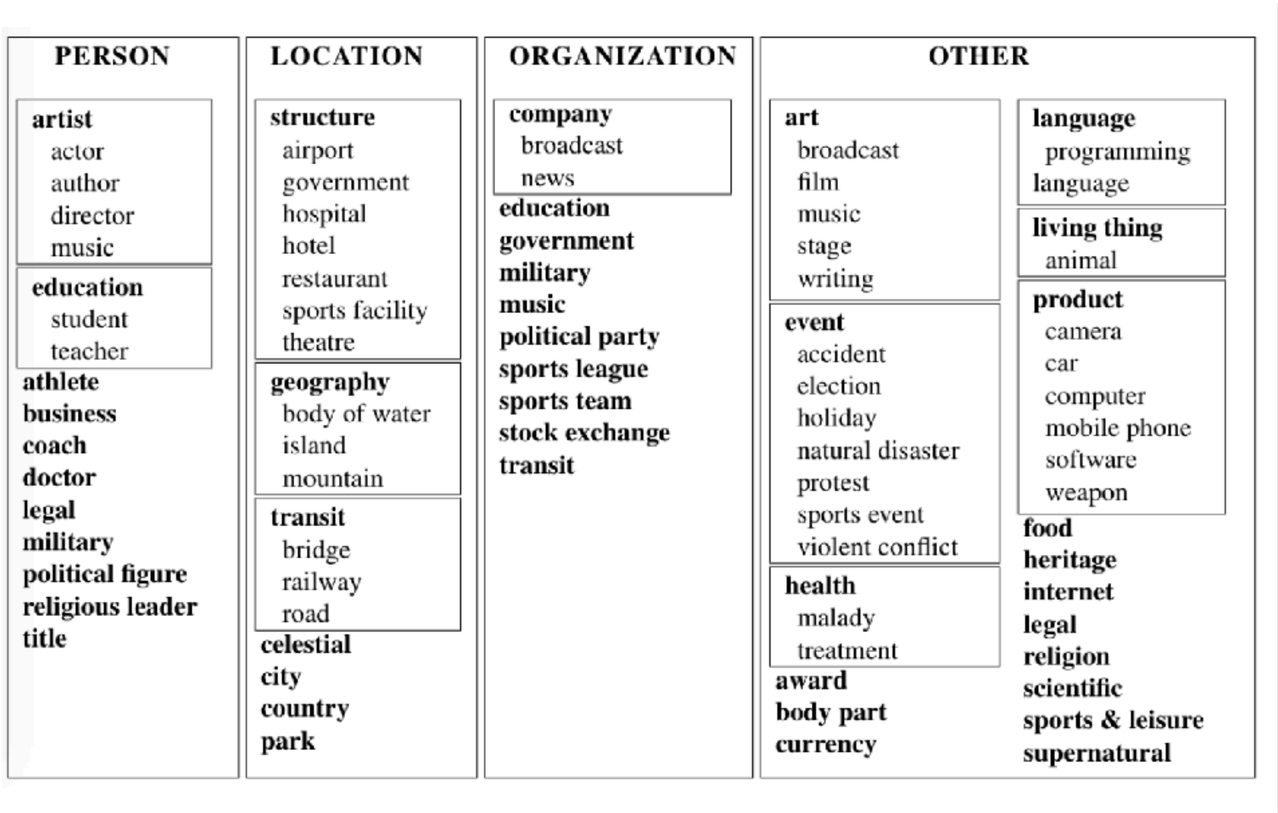
- a syntactic tree to build the dependencies, either using a parser or a PCFG/LFG grammar (both may take input from the previous step, which will provide the notion of verbal valence: *Person Browse Product*),

- for tied analyses, an interface with the user to request clarification.

Recent advances

Supersense inference

GFT (Google Fine-Grained) taxonomy



PERSON

- **artist**
  - actor
  - author
  - director
  - music
- **education**
  - student
  - teacher
- **athlete**
- **business**
- **coach**
- **doctor**
- **legal**
- **military**
- **political figure**
- **religious leader**
- **title**

LOCATION

- **structure**
  - airport
  - government
  - hospital
  - hotel
  - restaurant
  - sports facility
  - theatre
- **geography**
  - body of water
  - island
  - mountain
- **transit**
  - bridge
  - railway
  - road
- **celestial**
- **city**
- **country**
- **park**

ORGANIZATION

- **company**
  - broadcast
  - news
- **education**
- **government**
- **military**
- **music**
- **political party**
- **sports league**
- **sports team**
- **stock exchange**
- **transit**

OTHER

- **art**
  - broadcast
  - film
  - music
  - stage
  - writing
- **event**
  - accident
  - election
  - holiday
  - natural disaster
  - protest
  - sports event
  - violent conflict
- **health**
  - malady
  - treatment
- **award**
- **body part**
- **currency**
- **language**
  - programming
- language
- **living thing**
  - animal
- **product**
  - camera
  - car
  - computer
  - mobile phone
  - software
  - weapon
- **food**
- **heritage**
- **internet**
- **legal**
- **religion**
- **scientific**
- **sports & leisure**
- **supernatural**

Yogatama, Dani, Daniel Gillick, and Nevena Lazic. "Embedding Methods for Fine Grained Entity Type Classification." *ACL (2)*. 2015.

Supersense inference

FIGER taxonomy

person		doctor	**organization**		terrorist_organization		
actor		engineer	airline		government_agency		
architect		monarch	company		government		
artist		musician	educational_institution		political_party		
athlete		politician	fraternity_sorority		educational_department		
author		religious_leader	sports_league		military		
coach		soldier	sports_team		news_agency		
director		terrorist					
location		body_of_water	**product**		camera	**art**	written_work
city		island	engine		mobile_phone	film	newspaper
country		mountain	airplane		computer	play	music
county		glacier	car		software		
province		astral_body	ship		game	**event**	military_conflict
railway		cemetery	spacecraft		instrument	attack	natural_disaster
road		park	train		weapon	election	sports_event
bridge						protest	terrorist_attack
building	time		chemical_thing	website			
airport	color		biological_thing	broadcast_network			
dam	award		medical_treatment	broadcast_program			
hospital	educational_degree		disease	tv_channel			
hotel	title		symptom	currency			
library	law		drug	stock_exchange			
power_station	ethnicity		body_part	algorithm			
restaurant	language		living_thing	programming_language			
sports_facility	religion		animal	transit_system			
theater	god		food	transit_line			

Yogatama, Dani, Daniel Gillick, and Nevena Lazic. "Embedding Methods for Fine Grained Entity Type Classification." *ACL (2)*. 2015.

Supersense inference

```
FB = Freebase
T = OurTaxonomy[1]
owem = OurWordEmbeddingsModel
for every document d in D[2]:
    for every sent s in d:
        sentence_vector = None
        for every position, term, tag tuple in TermExtractor[3](s):
            if not sentence_vector:
                sentence_vector = vectorizeSentence(s, position)
            owem[ T[tag] ].update( sentence_vector )
```

[1] FB labels are manually mapped to fine-grained labels.
[2] D = 133,000 news documents.
[3] TermExtractor = parser + chunker + entity resolver that assigns Freebase types to entities.

Yogatama, Dani, Daniel Gillick, and Nevena Lazic. "Embedding Methods for Fine Grained Entity Type Classification." *ACL (2)*. 2015.

Supersense inference

FB = Freebase

T = OurTaxonomy[1]

owem =

for ever

for e

s

.

Feature	Description	Example
Head	The syntactic head of the mention phrase	"Obama"
Non-head	Each non-head word in the mention phrase	"Barack", "H."
Cluster	Word cluster id for the head word	"59"
Characters	Each character trigram in the mention head	":ob", "oba", "bam", "ama", "ma:"
Shape	The word shape of the words in the mention phrase	"Aa A. Aa"
Role	Dependency label on the mention head	"subj"
Context	Words before and after the mention phrase	"B:who", "A:first"
Parent	The head's lexical parent in the dependency tree	"picked"
Topic	The most likely topic label for the document	"politics"

sentence_vector = vectorizeSentence(*s, position*)

owem[T[*tag*]].update(sentence_vector)

[1] FB labels are manually mapped to fine-grained labels.
[2] D = 133,000 news documents.
[3] TermExtractor = parser + chunker + entity resolver that assigns Freebase types to entities.

Yogatama, Dani, Daniel Gillick, and Nevena Lazic. "Embedding Methods for Fine Grained Entity Type Classification." *ACL (2)*. 2015.

Supersense inference

Method	P	R	F1
FLAT	79.22	60.18	68.40
BINARY	80.05	62.20	70.01
WSABIE	**80.58**	66.20	72.68
K-WSABIE	80.11	**67.01**	**72.98**

Yogatama, Dani, Daniel Gillick, and Nevena Lazic. "Embedding Methods for Fine Grained Entity Type Classification." *ACL (2)*. 2015.

Fact-driven generation

W = Topic description, raw text about the topic, e.g. "*Rogers was born in Latrobe, Pennsylvania in 1928...*" (such as Wikipedia)

F = Facts associated with the topic (as triples from FreeBase), such as:

a42 = (Fred_Rogers, Place_of_Birth, Latrobe_Pennsylvania)
a83 = (Fred_Rogers, Year_of_Birth, 1928)
a0 = (Fred_Rogers, Topic_Itself, Fred_Rogers)

The corpus contains pairs $(W_k, F_k)_k^K$

Ahn, Sungjin, et al. "A neural knowledge language model." *arXiv preprint arXiv:1608.00318* (2016).

Fact-driven generation

Two sources of output:

Knowledge words O_a of a fact *a* is all words (o_{a1}, o_{a2},..., o_{aN}).
If selected, these words are copied to the output.

Global vocabulary *V* containing frequent words.
Words describing relationships (e.g., “married to”) are common and thus can be generated via the vocabulary *V* not via copy.

Ahn, Sungjin, et al. "A neural knowledge language model." *arXiv preprint arXiv:1608.00318* (2016).

Fact-driven generation

During training:

Simple string matching to map words in W to facts in F.

Since not all words are associated with a fact (e.g., words *like, is, a, the, have*), the *Not-a-Fact (NaF)* type is used to generate those words.

... *Fred_Rogers* was born in **Pennsylvania**...

t-4 *t-3* *t-2* *t-1* ***t***

$p(O \mid born, a, Place\_of\_Birth, o_{ax}) >$ (in the embeddings space)
$p(V \mid born, a, Place\_of\_Birth, o_{ax})$

Ahn, Sungjin, et al. "A neural knowledge language model." *arXiv preprint arXiv:1608.00318* (2016).

Fact-driven generation

During training:

Simple string matching to map words in W to facts in F.

Since not all words are associated with a fact (e.g., words *like, is, a, the, have*), the *Not-a-Fact (NaF)* type is used to generate those words.

... *Fred_Rogers* was born in **Pennsylvania**...

t-4 *t-3* *t-2* *t-1* ***t***

No other words are needed (RNN)

$p(O \mid born, a, Place\_of\_Birth, o_{ax}) > p(V \mid born, a, Place\_of\_Birth, o_{ax})$ (in the embeddings space)

Ahn, Sungjin, et al. "A neural knowledge language model." *arXiv preprint arXiv:1608.00318* (2016).

Fact-driven generation

During training:

Simple string matching to map words in W to facts in F.

Since not all words are associated with a fact (e.g., words *like, is, a, the, have*), the *Not-a-Fact (NaF)* type is used to generate those words.

... *Fred_Rogers* was born in **Pennsylvania**...

t-4 *t-3* *t-2* *t-1* ***t***

Currently fact activated

$p(O \mid born, a, Place\_of\_Birth, o_{ax}) >$
$p(V \mid born, a, Place\_of\_Birth, o_{ax})$

(in the embeddings space)

Ahn, Sungjin, et al. "A neural knowledge language model." *arXiv preprint arXiv:1608.00318* (2016).

Fact-driven generation

During training:

Simple string matching to map words in W to facts in F.

Since not all words are associated with a fact (e.g., words *like, is, a, the, have*), the *Not-a-Fact (NaF)* type is used to generate those words.

... *Fred_Rogers* was born in **Pennsylvania**...

t-4 *t-3* *t-2* *t-1* ***t***

$p(O \mid born, a, Place\_of\_Birth, o_{ax}) >$ (in the embeddings space)

$p(V \mid born, a, Place\_of\_Birth, o_{ax})$

Fact word belong to this topic (as positive example)

Ahn, Sungjin, et al. "A neural knowledge language model." *arXiv preprint arXiv:1608.00318* (2016).

Fact-driven generation

During inference:

$$P(a|h_t) = \frac{\exp(\mathbf{k}_{\text{fact}}^\top \mathbf{F}[a])}{\sum_{a' \in \mathcal{F}} \exp(\mathbf{k}_{\text{fact}}^\top \mathbf{F}[a'])},$$

$$a_t = \underset{a \in \mathcal{F}}{\operatorname{argmax}}\, P(a|h_t),$$

$$\mathbf{a}_t = \mathbf{F}[a_t].$$

Output the fact word associated with the relation that maximizes the similarity with the state at *t*

Ahn, Sungjin, et al. "A neural knowledge language model." *arXiv preprint arXiv:1608.00318* (2016).

Semantic trees

Google DeepMind's Neural Turing Machines...

Graves, Alex, Greg Wayne, and Ivo Danihelka. "Neural turing machines." arXiv preprint arXiv:1410.5401 (2014).

... applied to a Question and Answer task.

Yin, Pengcheng, et al. "Neural enquirer: Learning to query tables with natural language." *arXiv preprint arXiv:1512.00965* (2015).

Semantic trees

q = "*Which city hosted the longest game before the game in Beijing?*"

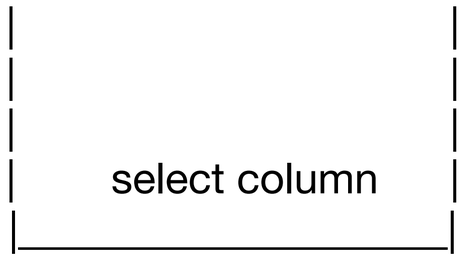
Yin, Pengcheng, et al. "Neural enquirer: Learning to query tables with natural language." *arXiv preprint arXiv:1512.00965* (2015).

Semantic trees

q = "*Which city hosted the longest game before the game in Beijing?*"

Which city hosted| *the longest game before the game in Beijing?*

retrieve (=SELECT) column whose title name's embedding
matches the embedding for input word 'city'



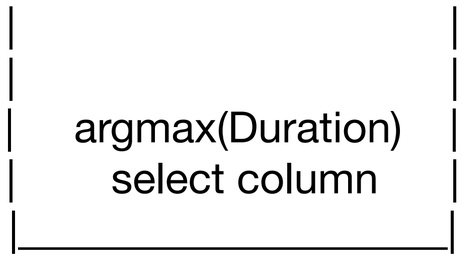
Yin, Pengcheng, et al. "Neural enquirer: Learning to query tables with natural language." *arXiv preprint arXiv:1512.00965* (2015).

Semantic trees

q = "*Which city hosted the longest game before the game in Beijing?*"

Which city hosted| *the longest game*| *before the game in Beijing?*

argmax of the column whose title name's embedding
matches embedding for input word 'longest'



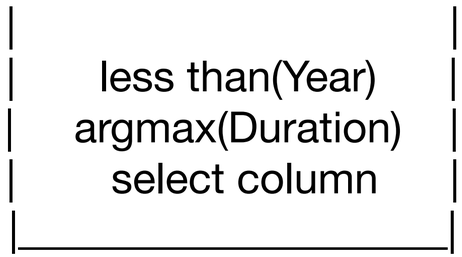
Yin, Pengcheng, et al. "Neural enquirer: Learning to query tables with natural language." *arXiv preprint arXiv:1512.00965* (2015).

Semantic trees

q = "*Which city hosted the longest game before the game in Beijing?*"

Which city hosted| *the longest game*| *before*| *the game in Beijing?*

less than over the column whose title name's embedding
matches embedding for input word 'before'



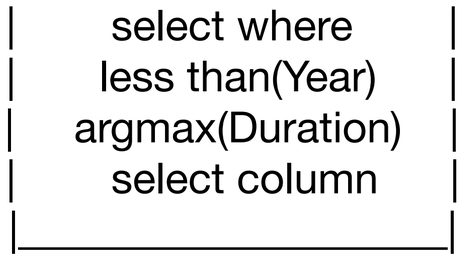
Yin, Pengcheng, et al. "Neural enquirer: Learning to query tables with natural language." *arXiv preprint arXiv:1512.00965* (2015).

Semantic trees

q = "*Which city hosted the longest game before the game in Beijing?*"

Which city hosted| *the longest game*| *before*| *the game in Beijing*|*?*

over all the rows, row containing the value whose embedding matches the embedding for input word 'Beijing'



Yin, Pengcheng, et al. "Neural enquirer: Learning to query tables with natural language." *arXiv preprint arXiv:1512.00965* (2015).

Semantic trees

q = "*Which city hosted the longest game before the game in Beijing?*"
KB = knowledge base consisting of tables that store facts
encoder = bi-directional Recurrent Neural Network
v = encoder(q)   # vector of the encoded query
kb_encoder = KB table encoder

Yin, Pengcheng, et al. "Neural enquirer: Learning to query tables with natural language." *arXiv preprint arXiv:1512.00965* (2015).

Semantic trees

q = "*Which city hosted the longest game before the game in Beijing?*"
KB = knowledge base consisting of tables that store facts
encoder = bi-directional Recurrent Neural Network
v = encoder(q) # vector of the encoded query
kb_encoder = KB table encoder

$$e_{mn} = \tanh(\mathbf{W} \cdot [\mathbf{L}[w_{mn}]; \mathbf{f}_n] + \mathbf{b})$$

f_n = embedding of the name of the *n*-th column
w_{mn} = embedding of the value in the *n*-th column of the *m*-th row
[x ; y] = vector concatenation
W = weights (what we will e.g. SGD out of the data)
b = bias

How the information on each DB table is encoded

Yin, Pengcheng, et al. "Neural enquirer: Learning to query tables with natural language." *arXiv preprint arXiv:1512.00965* (2015).

Semantic trees

q = "*Which city hosted the longest game before the game in Beijing?*"
KB = knowledge base consisting of tables that store facts
encoder = bi-directional Recurrent Neural Network
v = encoder(q) # vector of the encoded query
kb_encoder = KB table encoder

$$e_{mn} = \tanh(\mathbf{W} \cdot [\mathbf{L}[w_{mn}]; \mathbf{f}_n] + \mathbf{b})$$

f_n = embedding of the name of the *n*-th column
w_{mn} = embedding of the value in the *n*-th column of the *m*-th row
[x ; y] = vector concatenation
W = weights (what we will e.g. SGD out of the data)
b = bias

```
normalizeBetween0and1(
  penaltiesOrRewards *        # the weights that minimize the loss for this column
  [ {sydney... australia... kangaroo...} + {city... town... place...} ] +
  somePriorExpectations
)
```

Yin, Pengcheng, et al. "Neural enquirer: Learning to query tables with natural language." *arXiv preprint arXiv:1512.00965* (2015).

Semantic trees

q = "*Which city hosted the longest game before the game in Beijing?*"
KB = knowledge base consisting of tables that store facts
encoder = bi-directional Recurrent Neural Network
v = encoder(q) # vector of the encoded query
kb_encoder = KB table encoder

$$e_{mn} = \tanh(\mathbf{W} \cdot [\mathbf{L}[w_{mn}]; \mathbf{f}_n] + \mathbf{b})$$

f_n = embedding of the name of the *n*-th column
w_{mn} = embedding of the value in the *n*-th column of the *m*-th row
[x ; y] = vector concatenation
W = weights (what we will e.g. SGD out of the data)
b = bias

```
normalizeBetween0and1(
  penaltiesOrRewards *
  [ {sydney... australia... kangaroo...} + {city... town... place...} ] +
  somePriorExpectations
)
```

"*olympic games near kangaroos*" 👍

Yin, Pengcheng, et al. "Neural enquirer: Learning to query tables with natural language." *arXiv preprint arXiv:1512.00965* (2015).

Semantic trees

q = "*Which city hosted the longest game before the game in Beijing?*"
KB = knowledge base consisting of tables that store facts
encoder = bi-directional Recurrent Neural Network
v = encoder(q) # vector of the encoded query
kb_encoder = KB table encoder

$$e_{mn} = \tanh(\mathbf{W} \cdot [\mathbf{L}[w_{mn}]; \mathbf{f}_n] + \mathbf{b})$$

f_n = embedding of the name of the *n*-th column
w_{mn} = embedding of the value in the *n*-th column of the *m*-th row
[x ; y] = vector concatenation
W = weights (what we will e.g. SGD out of the data)
b = bias

```
normalizeBetween0and1(
  penaltiesOrRewards *
  [ {sydney... australia... kangaroo...} + {city... town... place...} ] +
  somePriorExpectations
)
```

"*olympic games near kangaroos*" 👍

Risk of false positives?
Already as probabilities!!

Yin, Pengcheng, et al. "Neural enquirer: Learning to query tables with natural language." *arXiv preprint arXiv:1512.00965* (2015).

Semantic trees

During training:

Olympic Games

City	Year	Duration
Barcelona	1992	15
Atlanta	1996	18
Sydney	2000	17
Athens	2004	14
Beijing	2008	16
London	2012	16
Rio de Janeiro	2016	18

Yin, Pengcheng, et al. "Neural enquirer: Learning to query tables with natural language." *arXiv preprint arXiv:1512.00965* (2015).

Semantic trees

Olympic Games

City	Year	Duration
Barcelona	1992	15
Atlanta	1996	18
Sydney	2000	17
Athens	2004	14
Beijing	2008	16
London	2012	16
Rio de Janeiro	2016	18

Which city hosted the 2012 Olympic Games?

Yin, Pengcheng, et al. "Neural enquirer: Learning to query tables with natural language." *arXiv preprint arXiv:1512.00965* (2015).

Semantic trees

Olympic Games

$T^{(i)}$ =

City	Year	Duration
Barcelona	1992	15
Atlanta	1996	18
Sydney	2000	17
Athens	2004	14
Beijing	2008	16
London	2012	16
Rio de Janeiro	2016	18

$y^{(i)}$ =

$Q^{(i)}$ = *Which city hosted the 2012 Olympic Games?*

RNN-encoded = *p(vector("column:city") | [vector("which"), vector("city")...])* >
p(vector("column:city") | [vector("city"), vector("which")...])

Yin, Pengcheng, et al. "Neural enquirer: Learning to query tables with natural language." *arXiv preprint arXiv:1512.00965* (2015).

Semantic trees

$Q^{(i)}$ =	*Which city hosted the 2012 Olympic Games?*
RNN-encoded =	*p(vector("column:city") | [vec("which"), vec("city")...])* >
Executor layer *L*	*p(vector("column:city") | [vec("city"), vec("which")...])*
Executor layer *L - 1*	*p(vector("column:year") | [... vec("2012"), vec("olympic") ...])* >
	p(vector("column:year") | [... vec("olympic"), vec("2012") ...])
Executor layer *L - 2*	*p(vector("FROM") | [... vec("olympic"), vec("games") ...])* >
	p(vector("FROM") | [... vec("games"), vec("olympic") ...])

As many as SQL operations

Yin, Pengcheng, et al. "Neural enquirer: Learning to query tables with natural language." *arXiv preprint arXiv:1512.00965* (2015).

Semantic trees

$Q^{(i)}$ =	*Which city hosted the 2012 Olympic Games?*
RNN-encoded =	*p(vector("column:city") | [vec("which"), vec("city")...])* >
Executor layer *L*	*p(vector("column:city") | [vec("city"), vec("which")...])*
Executor layer *L - 1*	*p(vector("column:year") | [... vec("2012"), vec("olympic") ...])* >
	p(vector("column:year") | [... vec("olympic"), vec("2012") ...])
Executor layer *L - 2*	*p(vector("FROM") | [... vec("olympic"), vec("games") ...])* >
	p(vector("FROM") | [... vec("games"), vec("olympic") ...])

Each executor applies the same set of weights to all rows (a it models a single operation)

Yin, Pengcheng, et al. "Neural enquirer: Learning to query tables with natural language." *arXiv preprint arXiv:1512.00965* (2015).

Semantic trees

$Q^{(i)}$ =	*Which city hosted the 2012 Olympic Games?*
RNN-encoded =	*p(vector("column:city") | [vec("which"), vec("city")...])* >
Executor layer *L*	*p(vector("column:city") | [vec("city"), vec("which")...])*
Executor layer *L - 1*	*p(vector("column:year") | [... vec("2012"), vec("olympic") ...])* >
	p(vector("column:year") | [... vec("olympic"), vec("2012") ...])
Executor layer *L - 2*	*p(vector("FROM") | [... vec("olympic"), vec("games") ...])* >
	p(vector("FROM") | [... vec("games"), vec("olympic") ...])

Before, After **are just temporal versions of < and > .**
Neural Turing Machines learn to sort accordingly.

Yin, Pengcheng, et al. "Neural enquirer: Learning to query tables with natural language." *arXiv preprint arXiv:1512.00965* (2015).

Semantic trees

$Q^{(i)}$ = *Which city hosted the 2012 Olympic Games?*

RNN-encoded = *p(vector("column:city") | [vec("which"), vec("city")...])* >

Executor layer *L* *p(vector("column:city") | [vec("city"), vec("which")...])*

Executor layer *L - 1* *p(vector("column:year") | [... vec("2012"), vec("olympic") ...])* >

p(vector("column:year") | [... vec("olympic"), vec("2012") ...])

Executor layer *L - 2* *p(vector("FROM") | [... vec("olympic"), vec("games") ...])* >

p(vector("FROM") | [... vec("games"), vec("olympic") ...])

Evaluate a candidate:

Candidate vector at layer *L* for row *m* of our database

$$\mathbf{r}_m^\ell = f_{\mathrm{R}}^\ell(\mathcal{R}_m, \mathcal{F}_\mathcal{T}, \mathbf{q}, \mathcal{M}^{\ell-1}) = \sum_{n=1}^{N} \tilde{\omega}(\mathbf{f}_n, \mathbf{q}, \mathbf{g}^{\ell-1})\mathbf{e}_{mn}$$

Memory layer with the last output vector generated by executor *L - 1*

Yin, Pengcheng, et al. "Neural enquirer: Learning to query tables with natural language." *arXiv preprint arXiv:1512.00965* (2015).

Semantic trees

$Q^{(i)}$ = *Which city hosted the 2012 Olympic Games?*

RNN-encoded = *p(vector("column:city") | [vec("which"), vec("city")...])* >

Executor layer *L* *p(vector("column:city") | [vec("city"), vec("which")...])*

Executor layer *L - 1* *p(vector("column:year") | [... vec("2012"), vec("olympic") ...])* >

p(vector("column:year") | [... vec("olympic"), vec("2012") ...])

Executor layer *L - 2* *p(vector("FROM") | [... vec("olympic"), vec("games") ...])* >

p(vector("FROM") | [... vec("games"), vec("olympic") ...])

As an attention mechanism, cumulatively weight the vector by the output weights from executor *L - 1*

Evaluate a candidate:

Row *m*, R = our table

$$\mathbf{r}_m^\ell = f_R^\ell(\mathcal{R}_m, \mathcal{F}_\mathcal{T}, \mathbf{q}, \mathcal{M}^{\ell-1}) = \sum_{n=1}^{N} \tilde{\omega}(\mathbf{f}_n, \mathbf{q}, \mathbf{g}^{\ell-1})\mathbf{e}_{mn}$$

Set of columns

Yin, Pengcheng, et al. "Neural enquirer: Learning to query tables with natural language." *arXiv preprint arXiv:1512.00965* (2015).

Semantic trees

$Q^{(i)}$ = *Which city hosted the 2012 Olympic Games?*

RNN-encoded = *p(vector("column:city") | [vec("which"), vec("city")...])* >

Executor layer *L* *p(vector("column:city") | [vec("city"), vec("which")...])*

Executor layer *L - 1* *p(vector("column:year") | [... vec("2012"), vec("olympic") ...])* >

p(vector("column:year") | [... vec("olympic"), vec("2012") ...])

Executor layer *L - 2* *p(vector("FROM") | [... vec("olympic"), vec("games") ...])* >

p(vector("FROM") | [... vec("games"), vec("olympic") ...])

This allows the system to restrict the reference at each step.

Evaluate a candidate:

Row *m*, R = our table

$$\mathbf{r}_m^\ell = f_{\mathrm{R}}^\ell(\mathcal{R}_m, \mathcal{F}_\mathcal{T}, \mathbf{q}, \mathcal{M}^{\ell-1}) = \sum_{n=1}^{N} \tilde{\omega}(\mathbf{f}_n, \mathbf{q}, \mathbf{g}^{\ell-1})\mathbf{e}_{mn}$$

Set of columns

Yin, Pengcheng, et al. "Neural enquirer: Learning to query tables with natural language." *arXiv preprint arXiv:1512.00965* (2015).

Semantic trees

$Q^{(i)}$ = *Which city hosted the 2012 Olympic Games?*

RNN-encoded = *p(vector("column:city") | [vec("which"), vec("city")...]) > p(vector("column:city") | [vec("city"), vec("which")...])*

Executor layer *L*

Executor layer *L - 1*

p(vector("column:year") | [... vec("2012"), vec("olympic") ...]) > p(vector("column:year") | [... vec("olympic"), vec("2012") ...])

Executor layer *L - 2*

p(vector("FROM") | [... vec("olympic"), vec("games") ...]) > p(vector("FROM") | [... vec("games"), vec("olympic") ...])

As we weight the vector in the direction of each successive operation, it gets closer to any applicable answer in our DB.

Evaluate a candidate:

Row *m*, R = our table

$$\mathbf{r}_m^\ell = f_{\mathrm{R}}^\ell(\mathcal{R}_m, \mathcal{F}_\mathcal{T}, \mathbf{q}, \mathcal{M}^{\ell-1}) = \sum_{n=1}^{N} \tilde{\omega}(\mathbf{f}_n, \mathbf{q}, \mathbf{g}^{\ell-1})\mathbf{e}_{mn}$$

Set of columns

Yin, Pengcheng, et al. "Neural enquirer: Learning to query tables with natural language." *arXiv preprint arXiv:1512.00965* (2015).

Semantic trees

	MIXTURED-25K			
	SEMPRE	N2N	SbS	N2N - OOV
SELECT_WHERE	93.8%	96.2%	99.7%	90.3%
SUPERLATIVE	97.8%	98.9%	99.5%	98.2%
WHERE_SUPERLATIVE	34.8%	80.4%	94.3%	79.1%
NEST	34.4%	60.5%	92.1%	57.7%
Overall Acc.	65.2%	84.0%	96.4%	81.3%

SbS is N2N with **bias.**

SEMPRE is a toolkit for training semantic parsers, which map natural language utterances to denotations (answers) via intermediate logical forms. Here's an example for querying databases. https://nlp.stanford.edu/software/sempre/

Yin, Pengcheng, et al. "Neural enquirer: Learning to query tables with natural language." *arXiv preprint arXiv:1512.00965* (2015).

Semantic trees

	MIXTURED-25K			
	SEMPRE	N2N	SbS	N2N - OOV
SELECT_WHERE	93.8%	96.2%	99.7%	90.3%
SUPERLATIVE	97.8%	98.9%	99.5%	98.2%
WHERE_SUPERLATIVE	34.8%	80.4%	94.3%	79.1%
NEST	34.4%	60.5%	92.1%	57.7%
Overall Acc.	65.2%	84.0%	96.4%	81.3%

SbS is N2N with **bias.**

SEMPRE is a toolkit for training semantic parsers, which map natural language utterances to denotations (answers) via intermediate logical forms. Here's an example for querying databases. https://nlp.stanford.edu/software/sempre/

Yin, Pengcheng, et al. "Neural enquirer: Learning to query tables with natural language." *arXiv preprint arXiv:1512.00965* (2015).

Semantic trees

	MIXTURED-25K			
	SEMPRE	N2N	SbS	N2N - OOV
SELECT_WHERE	93.8%	96.2%	99.7%	90.3%
SUPERLATIVE	97.8%	98.9%	99.5%	98.2%
WHERE_SUPERLATIVE	34.8%	80.4%	94.3%	79.1%
NEST	34.4%	60.5%	92.1%	57.7%
Overall Acc.	65.2%	84.0%	96.4%	81.3%

SbS is N2N with **bias.**

SEMPRE is a toolkit for training semantic parsers, which map natural language utterances to denotations (answers) via intermediate logical forms. Here's an example for querying databases. https://nlp.stanford.edu/software/sempre/

Yin, Pengcheng, et al. "Neural enquirer: Learning to query tables with natural language." *arXiv preprint arXiv:1512.00965* (2015).

Re-entry

Task: given a set of facts or a story, answer questions on those facts in a logically consistent way.

Problem: Theoretically, it could be achieved by a language modeler such as a recurrent neural network (RNN).

However, their memory (encoded by hidden states and weights) is typically too small, and is not compartmentalized enough to accurately remember facts from the past (knowledge is compressed into dense vectors).

RNNs are known to have difficulty in memorization tasks, i.e., outputting the same input sequence they have just read.

Weston, Jason, Sumit Chopra, and Antoine Bordes. "Memory networks." *arXiv preprint arXiv:1410.3916* (2014).

Re-entry

Task: given a set of facts or a story, answer questions on those facts in a logically consistent way.

Jason went to the kitchen.	**jason go kitchen**
Jason picked up the milk.	**jason get milk**
Jason travelled to the office.	**jason go office**
Jason left the milk there.	**jason drop milk**
Jason went to the bathroom.	**jason go bathroom**
Where is the milk?	**milk?**

Weston, Jason, Sumit Chopra, and Antoine Bordes. "Memory networks." *arXiv preprint arXiv:1410.3916* (2014).

Re-entry

Memory networks

At the reading stage...

F = set of facts (e.g. recent context of conversation; sentences)
M = memory
for fact *f* in F: **# any linguistic pre-processing may apply**
    store *f* in the next available memory slot of M

Weston, Jason, Sumit Chopra, and Antoine Bordes. "Memory networks."
arXiv preprint arXiv:1410.3916 (2014).

Re-entry

Memory networks

At the inference stage...

F = set of facts (e.g. recent context of conversation; sentences)
M = memory
O = inference module (oracle)
x = input
q = query about our known facts
k = number of candidate facts to be retrieved
m = a supporting fact (typically, the one that maximizes support)

$$o_2 = O_2(x, \mathbf{m}) = \arg\max_{i=1,\ldots,N} s_O([x, \mathbf{m}_{o_1}], \mathbf{m}_i)$$

Weston, Jason, Sumit Chopra, and Antoine Bordes. "Memory networks."
arXiv preprint arXiv:1410.3916 (2014).

Re-entry

Memory networks

At the inference stage...

F = set of facts (e.g. recent context of conversation; sentences)
M = memory
O = inference module (oracle)
x = input
q = query about our known facts
k = number of candidate facts to be retrieved
m = a supporting fact (typically, the one that maximizes support)

$$o_2 = O_2(x, \mathbf{m}) = \arg\max_{i=1,\ldots,N} s_O([x, \mathbf{m}_{o_1}], \mathbf{m}_i)$$

jason go kitchen
jason get milk
jason go office
jason drop milk
jason go bathroom

Weston, Jason, Sumit Chopra, and Antoine Bordes. "Memory networks."
arXiv preprint arXiv:1410.3916 (2014).

Re-entry

Memory networks

At the inference stage, t_1...

milk, *[ ]*

$$o_2 = O_2(x, \mathbf{m}) = \arg\max_{i=1,\ldots,N} s_O([x, \mathbf{m}_{o_1}], \mathbf{m}_i)$$

milk

jason go kitchen
jason get milk
jason go office
jason drop milk
jason go bathroom

Candidate space for k = 2

jason go kitchen
jason get milk
jason go office
jason drop milk
jason go bathroom

argmax over that space

Weston, Jason, Sumit Chopra, and Antoine Bordes. "Memory networks."
arXiv preprint arXiv:1410.3916 (2014).

Re-entry

Memory networks

At the inference stage, t_2...

milk, jason, drop

$$o_2 = O_2(x, \mathbf{m}) = \arg\max_{i=1,\ldots,N} s_O([x, \mathbf{m}_{o_1}], \mathbf{m}_i)$$

milk

jason go kitchen
jason get milk
jason go office
jason drop milk
jason go bathroom

Candidate space for k = 2

jason go kitchen
jason get milk
jason go office
jason drop milk
jason go bathroom

argmax over that space

Weston, Jason, Sumit Chopra, and Antoine Bordes. "Memory networks."
arXiv preprint arXiv:1410.3916 (2014).

Re-entry

Memory networks

At the inference stage, t_2...

$$o_2 = O_2(x, \mathbf{m}) = \arg\max_{i=1,\ldots,N} s_O([x, \mathbf{m}_{o_1}], \mathbf{m}_i)$$

jason go kitchen
jason get milk
jason go office
jason drop milk
jason go bathroom

jason go kitchen
jason get milk
jason go office
jason drop milk
jason go bathroom

Why not?

jason, get, milk = does not contribute as much new information as *jason go* ***office***

Weston, Jason, Sumit Chopra, and Antoine Bordes. "Memory networks." *arXiv preprint arXiv:1410.3916* (2014).

Re-entry

Memory networks

Training is performed with a margin ranking loss and stochastic gradient descent (SGD).

Simple model.

Easy to integrate with any pipeline already working with structured facts (e.g. Neural Inquirer). Adds a temporal dimension to it.

All previous research in coreference resolution applies.

Answering the question about the location of the milk requires comprehension of the actions *picked up* and *left*.

Weston, Jason, Sumit Chopra, and Antoine Bordes. "Memory networks."
arXiv preprint arXiv:1410.3916 (2014).

Re-entry

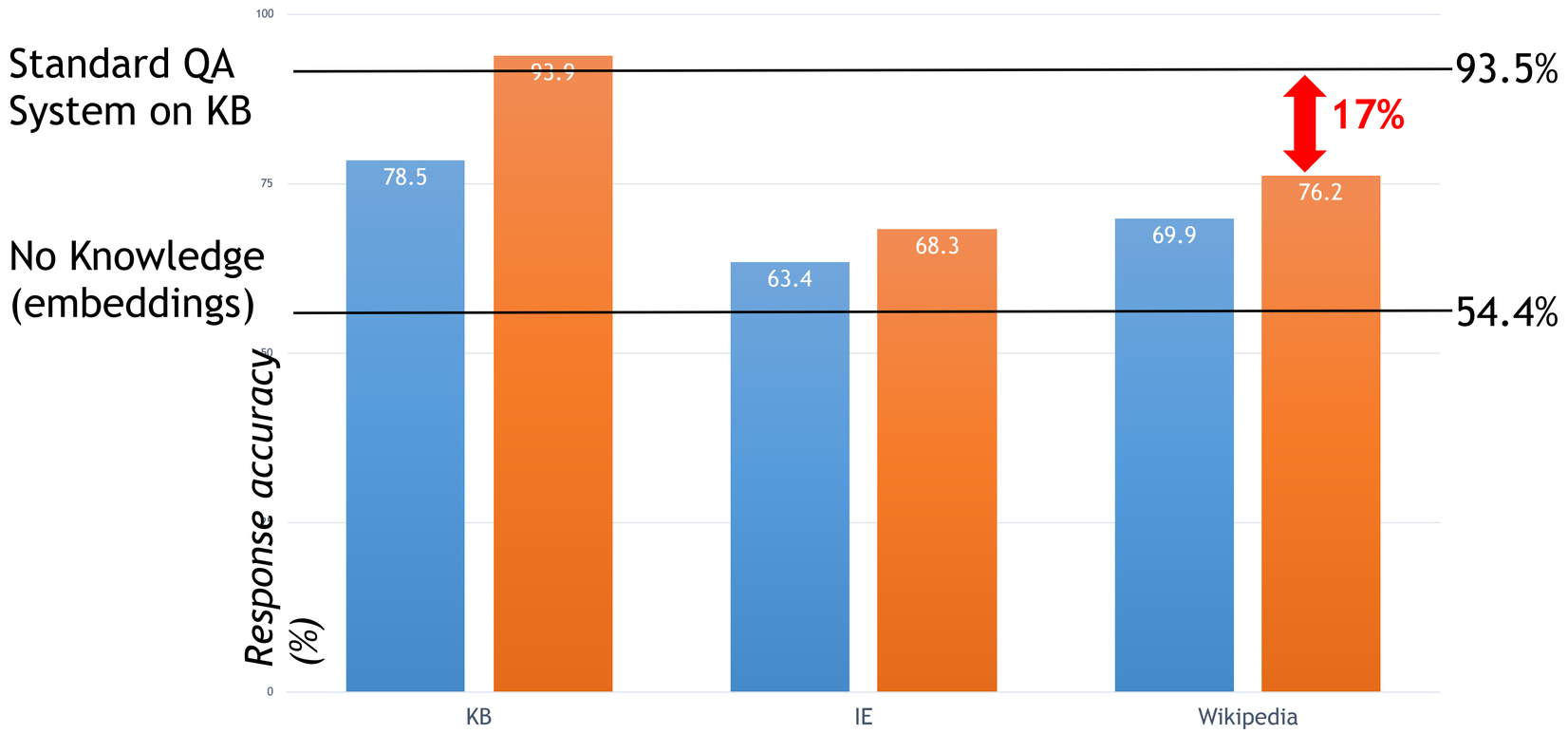
Memory networks

	QACNN	CBT-NE	CBT-CN	Date
	69.4	66.6	63.0	
Kadlec, Rudolf, et al. "Text understanding with the attention sum reader network." *arXiv preprint arXiv: 1603.01547* (2016).	75.4	71.0	68.9	4 Mar 16
Sordoni, Alessandro, et al. "Iterative alternating neural attention for machine reading." *arXiv preprint arXiv: 1606.02245* (2016).	76.1	72.0	71.0	7 Jun 16
Trischler, Adam, et al. "Natural language comprehension with the epireader." *arXiv preprint arXiv:1606.02270* (2016).	74.0	71.8	70.6	7 Jun 16
Dhingra, Bhuwan, et al. "Gated-attention readers for text comprehension." *arXiv preprint arXiv:1606.01549* (2016).	77.4	71.9	69.0	5 Jun 16

Weston, Jason, Sumit Chopra, and Antoine Bordes. "Memory networks." *arXiv preprint arXiv:1410.3916* (2014).

Source: Antoine Bordes, Facebook AI Research, LXMLS Lisbon July 28, 2016

Results on MovieQA



Memory Networks

Key-Value Memory Networks

Conclusions

Summary of *desiderata*

Enable semantic parsing using a neuralized architecture with respect to some knowledge base serving as ground truth.

Use memory networks to effectively keep track of the entities throughout the conversation and ensure logical consistency.

Use a modular neuralized architecture to compute answers for unseen questions dynamically.

Extend the neuralized architecture's domain by integrating support for supersenses. Enable inference at training and test time.

Extend memory networks by integrating support for supersenses. Enable inference at training and test time.

Summary of *desiderata*

Remember?

Conversational technology should really be about

- dynamically finding the best possible way to browse a large repository of information/actions.
- find the shortest path to any relevant action or piece of information (to avoid the plane dashboard effect).
- surfacing implicit data in unstructured content ("bocadillo de calamares in Madrid"). Rather than going open-domain, taking the closed-domain and going deep into it.

thanks!

Macy's On-Call

Today 09:57 AM

We've got what you're looking for!
Head to Floor 1 near Men's Collections
For more info on colors, sizes & more, click the image below.

Michael Kors Men's Leather Rac...