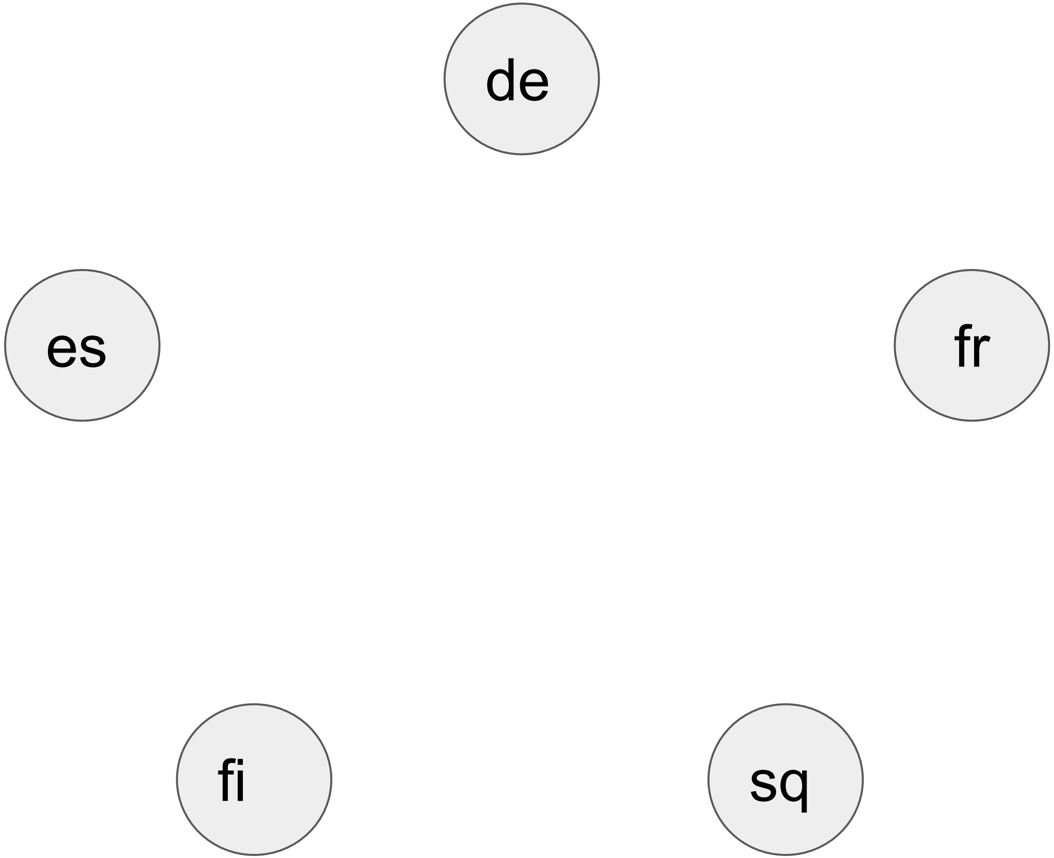
Multilinguality and MT

Jörg Tiedemann
Raúl Vázquez
Timothee Mickus

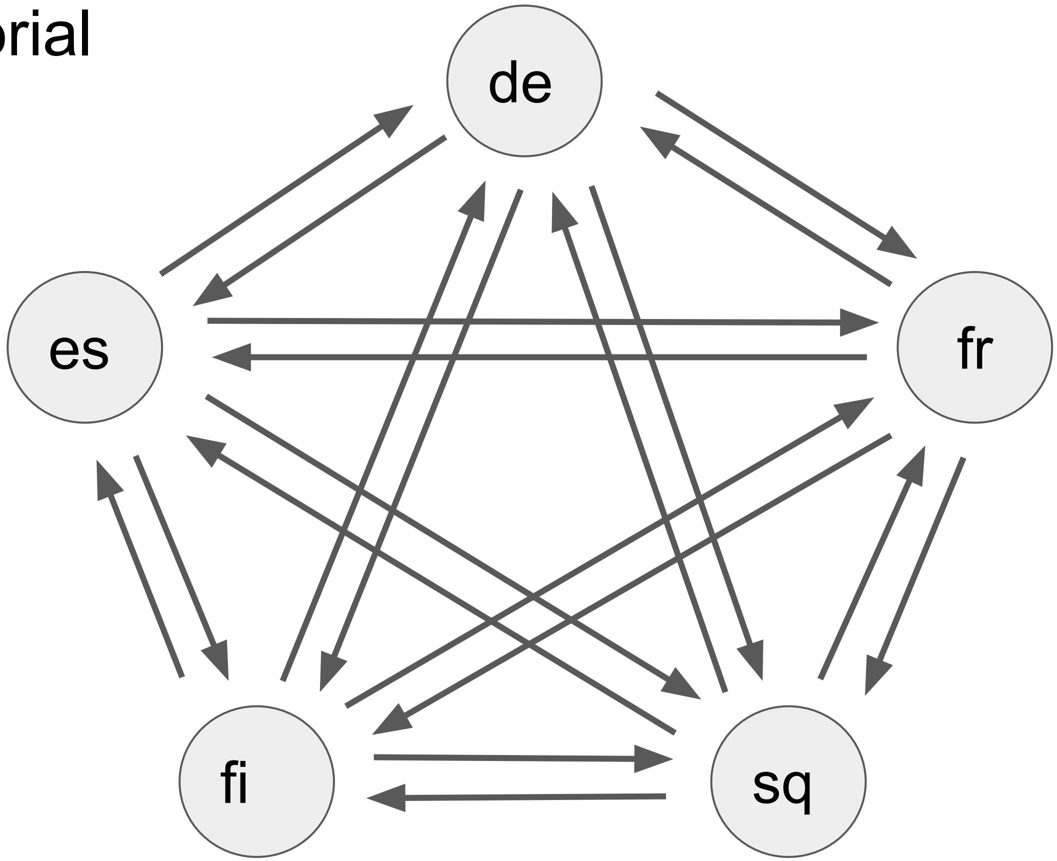
Outline

- Why?
  - The blessings of multilinguality
  - The curse of multilinguality
- How?
  - The MAMMOTH framework
  - Parameter sharing & modularity
  - Scaling up & parallelization
- Get involved (at the MT marathon and beyond)

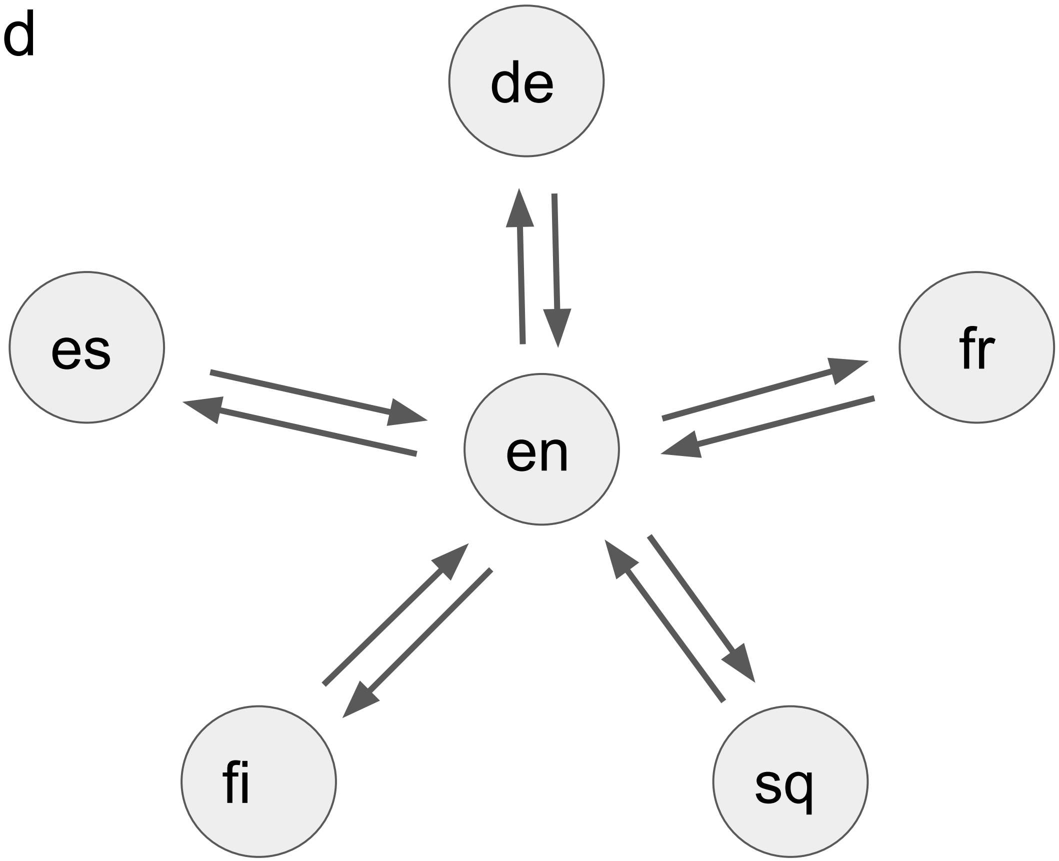
Why?



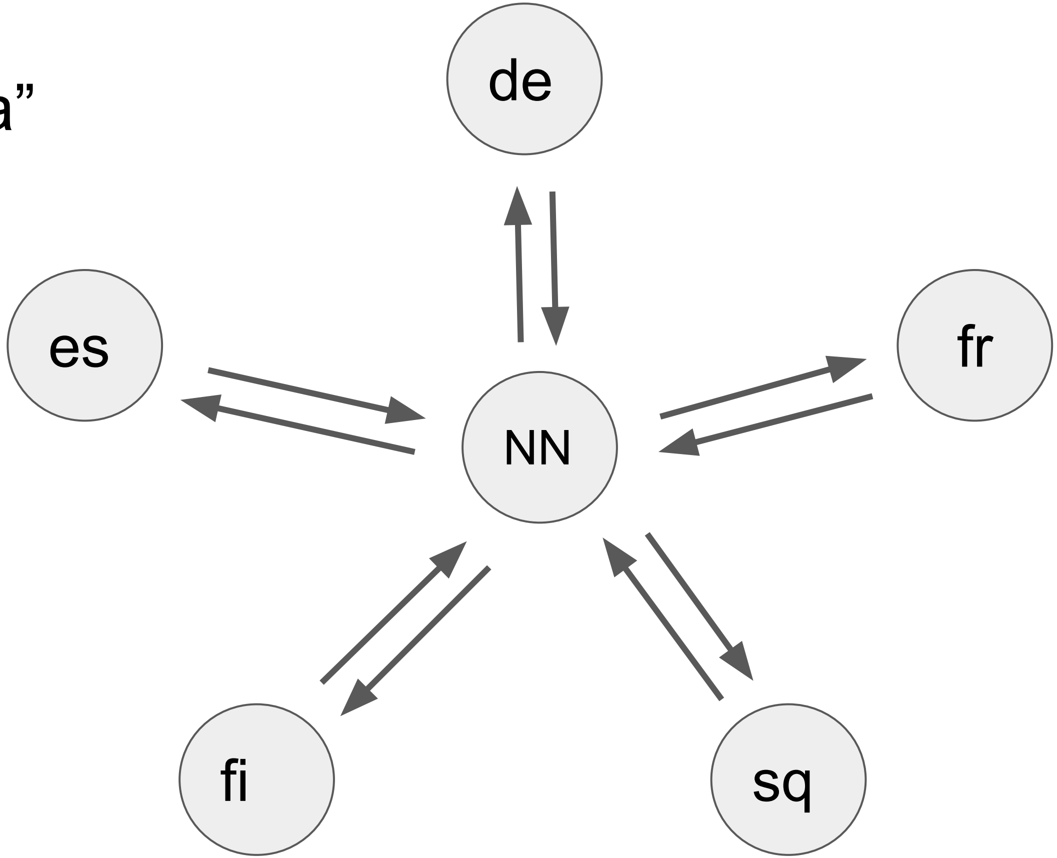
Combinatorial explosion



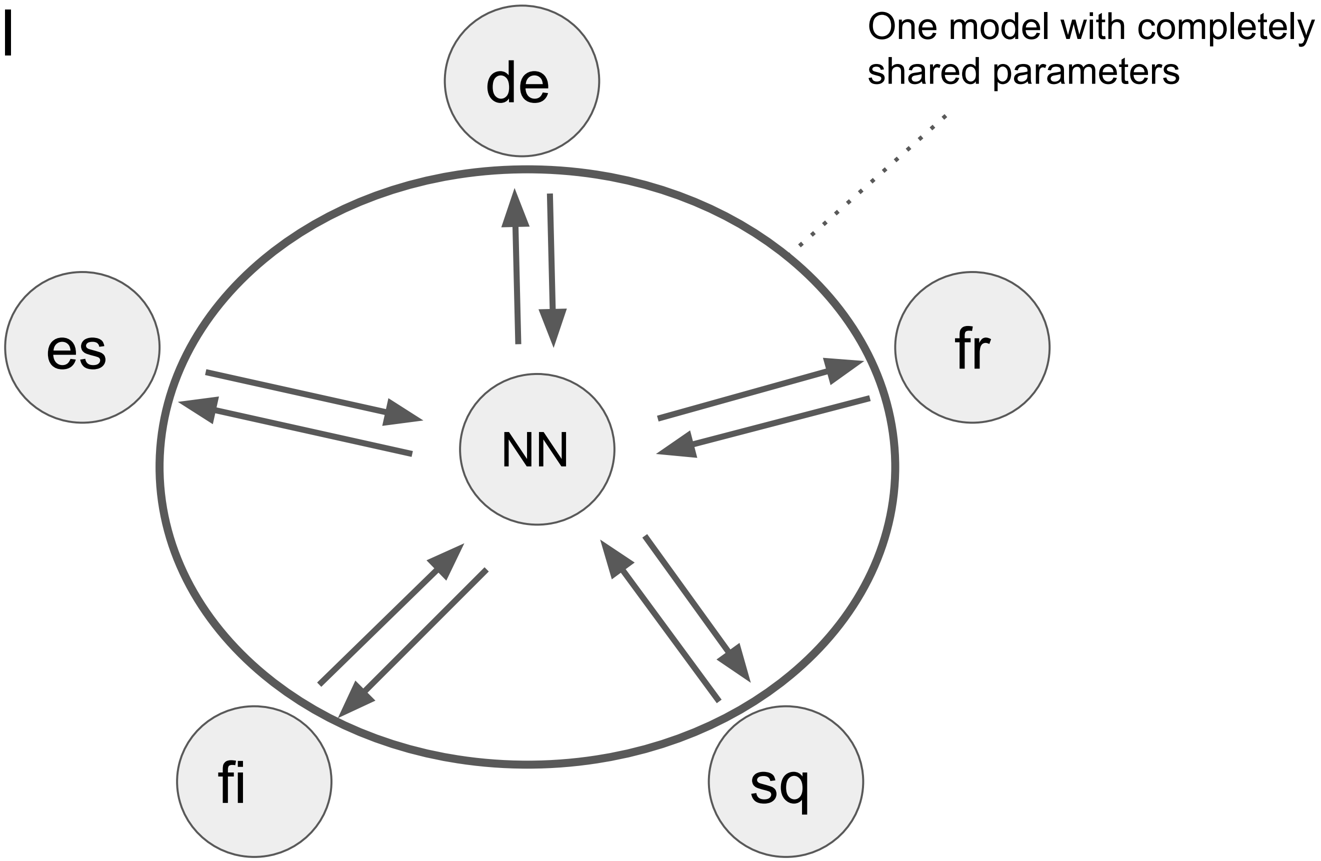
Pivot-based approach



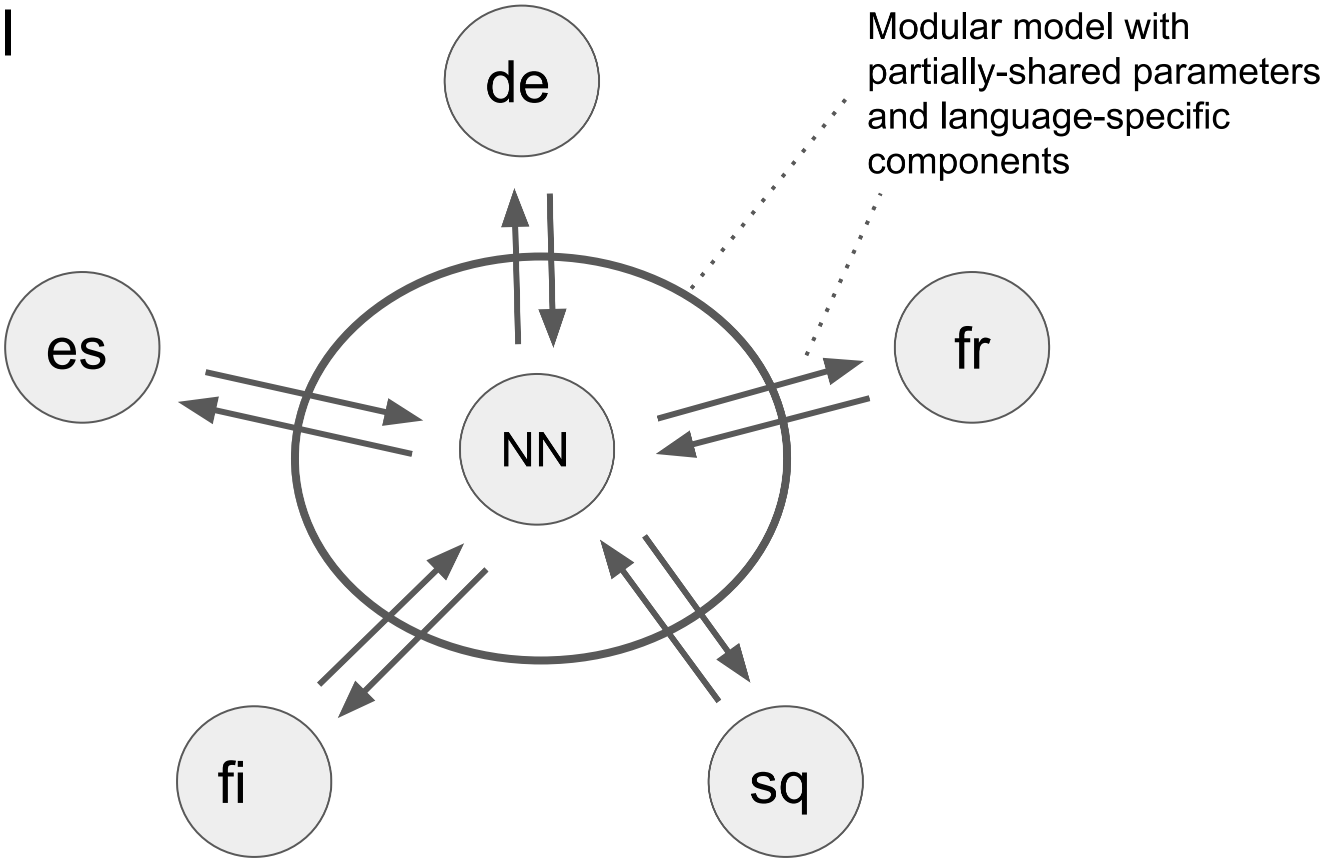
Neural
"interlingua"



Multilingual NMT



Multilingual NMT

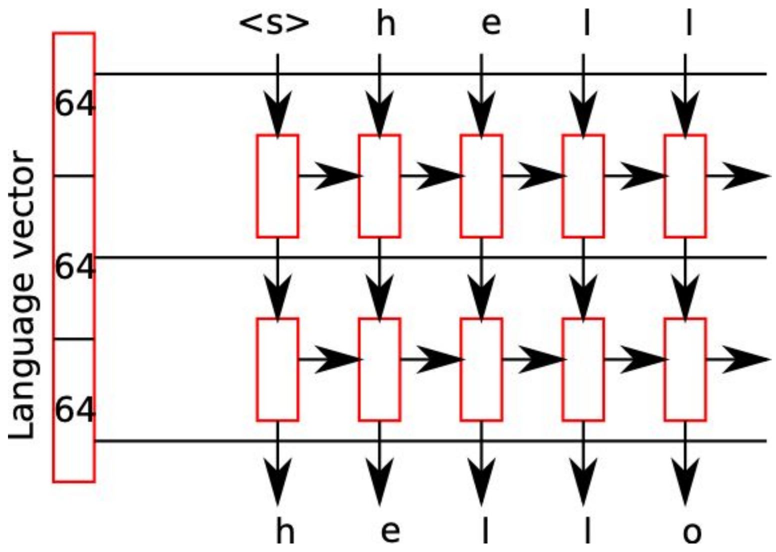


The Blessings of Multilinguality

The language continuum and language embeddings

Back in 2016:

1303 Bible translations
into 990 languages



Continuous multilinguality with language vectors

Robert Östling
Department of Linguistics*
Stockholm University
robert@ling.su.se

Jörg Tiedemann
Department of Modern Languages
University of Helsinki
jorg.tiedemann@helsinki.fi

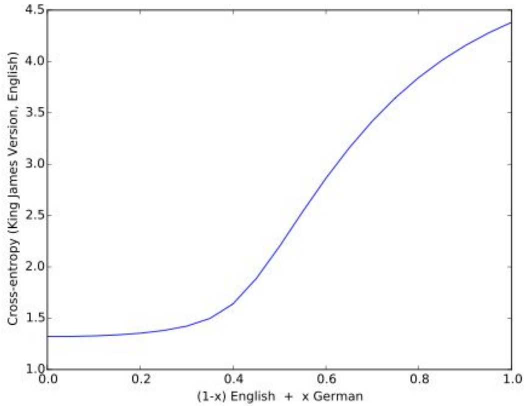
Abstract

Most existing models for multilingual natural language processing (NLP) treat language as a discrete category, and make predictions for either one language or the other. In contrast, we propose using continuous vector representations of language. We show that these can be learned

separate model for each language. This presupposes large quantities of monolingual data in each of the languages that needs to be covered and each model with its parameters is completely independent of any of the other models.

We propose instead to use a single model with real-valued vectors to indicate the language used, and to train this model with a large number of languages. We thus get a language model whose

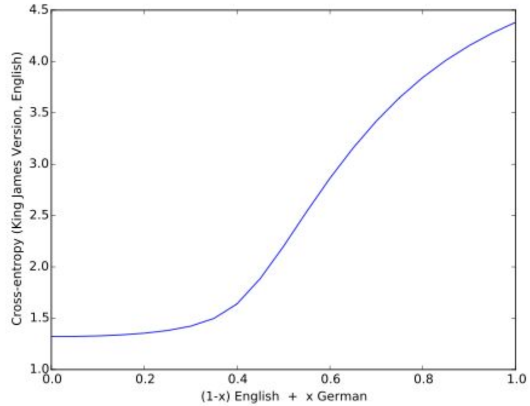
Continuous multilinguality with language vectors

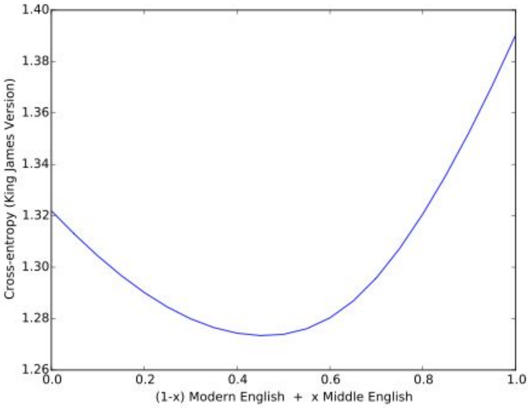


Interpolating between the English language vector and the German language vector

(cross-entropy for English)

Continuous multilinguality with language vectors

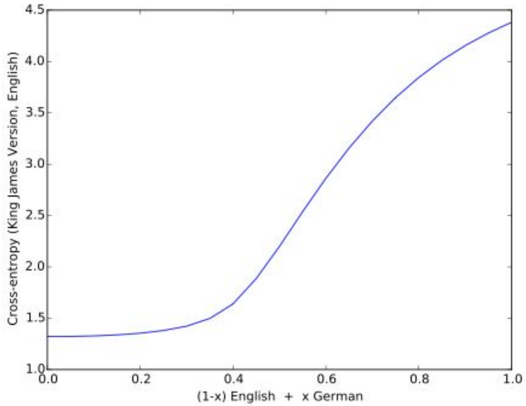


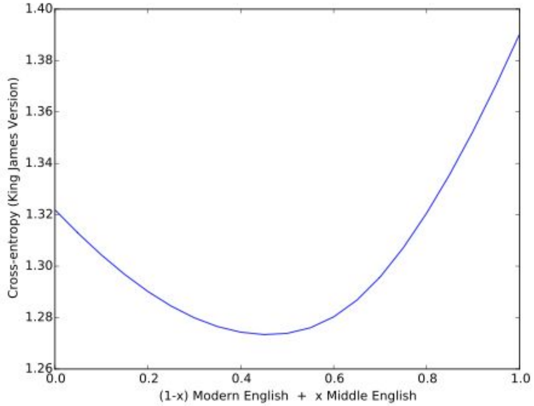


Interpolating between
Modern English and
Middle English

(cross-entropy for English from
the 17th century)

Continuous multilinguality with language vectors



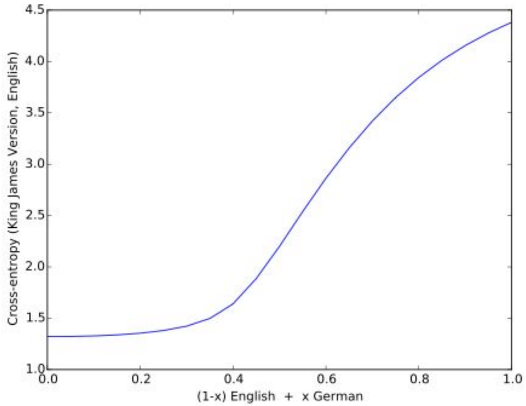


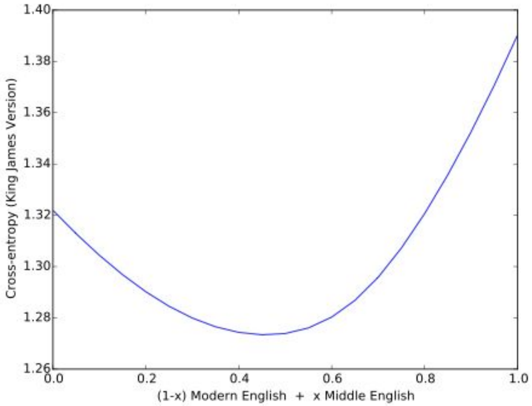
%	Random sample (temperature parameter $\tau = 0.5$)
30	and thei schulen go in to alle these thingis, and schalt endure bothe in the weie
40	and there was a certaine other person who was called in a dreame that he went into a mountaine.
44	and the second sacrifice, and the father, and the prophet, shall be given to it.
48	and god sayd, i am the light of the world, and the powers of the enemies of the most high god may find first for many.
50	but if there be some of the seruants, and to all the people, and the angels of god, and the prophets
52	then he came to the gate of the city, and the bread was to be brought
56	therefore, behold, i will lose the sound of my soul, and i will not fight it into the land of egypt
60	and the man whom the son of man is born of god, so have i therefore already sent to the good news of christ.

middle English

modern English

Continuous multilinguality with language vectors





Control text generation with language embeddings:

turn on Swedish:

och jehova sade till honom : ” jehova har sagt , och jag skall ...

turn on German:

und er sprach zu ihnen : siehe , ich bin der herr

mix Swedish and German:

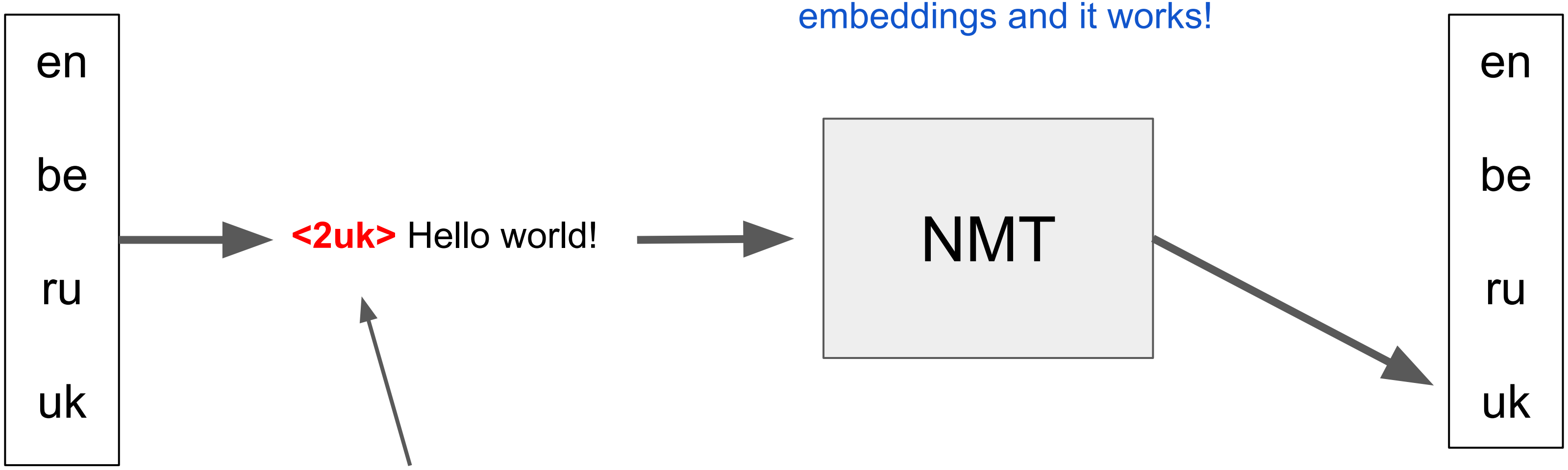
vocken ånner vocken ånnen söhenöckenföcken ...

average of Scandinavian languages:

og han sa til herrens : " han skal vitnaðus til herrens hjárt

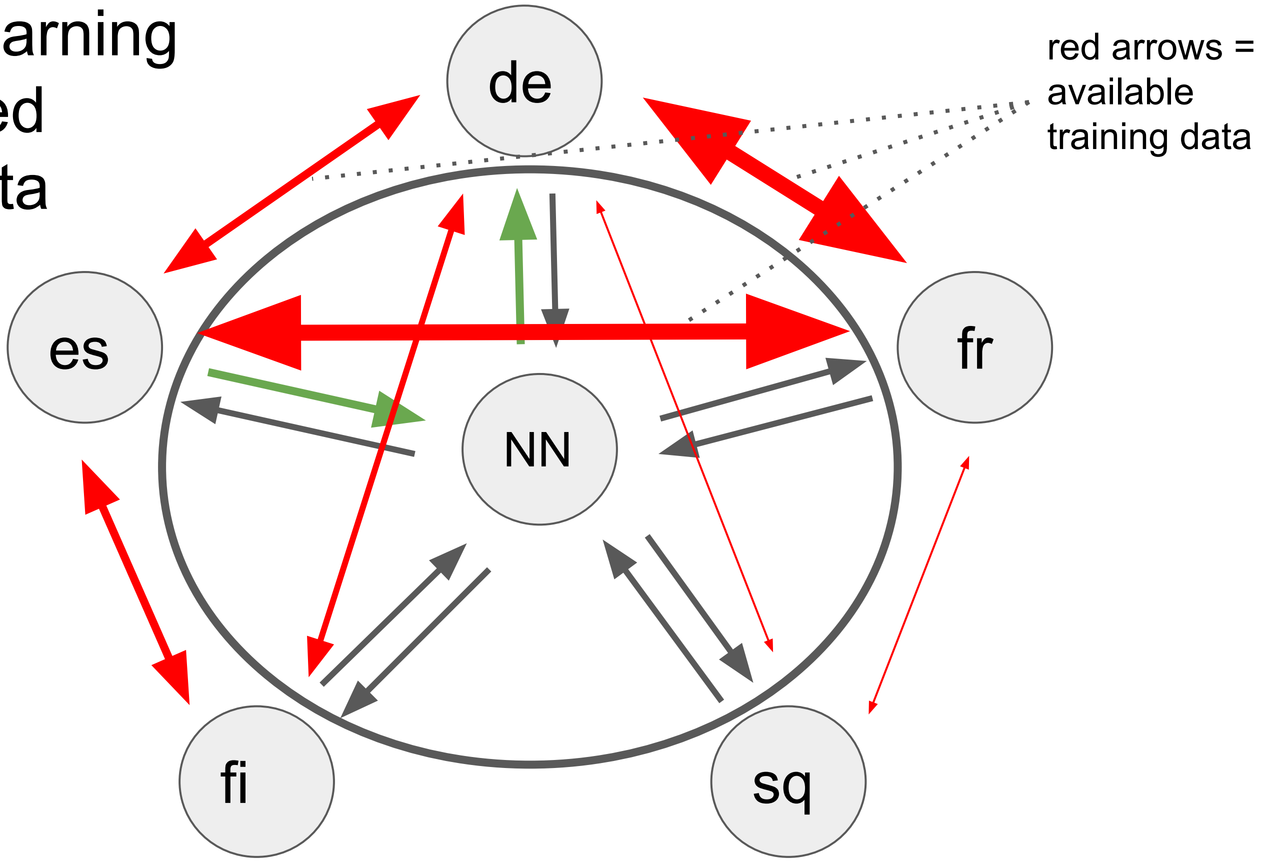
At the same time: Johnson et al. for multilingual NMT

Same idea about learning language embeddings and it works!



Add a special token that tells the system to translate into a particular language.
(early type of “prompting”)

Transfer learning with skewed training data

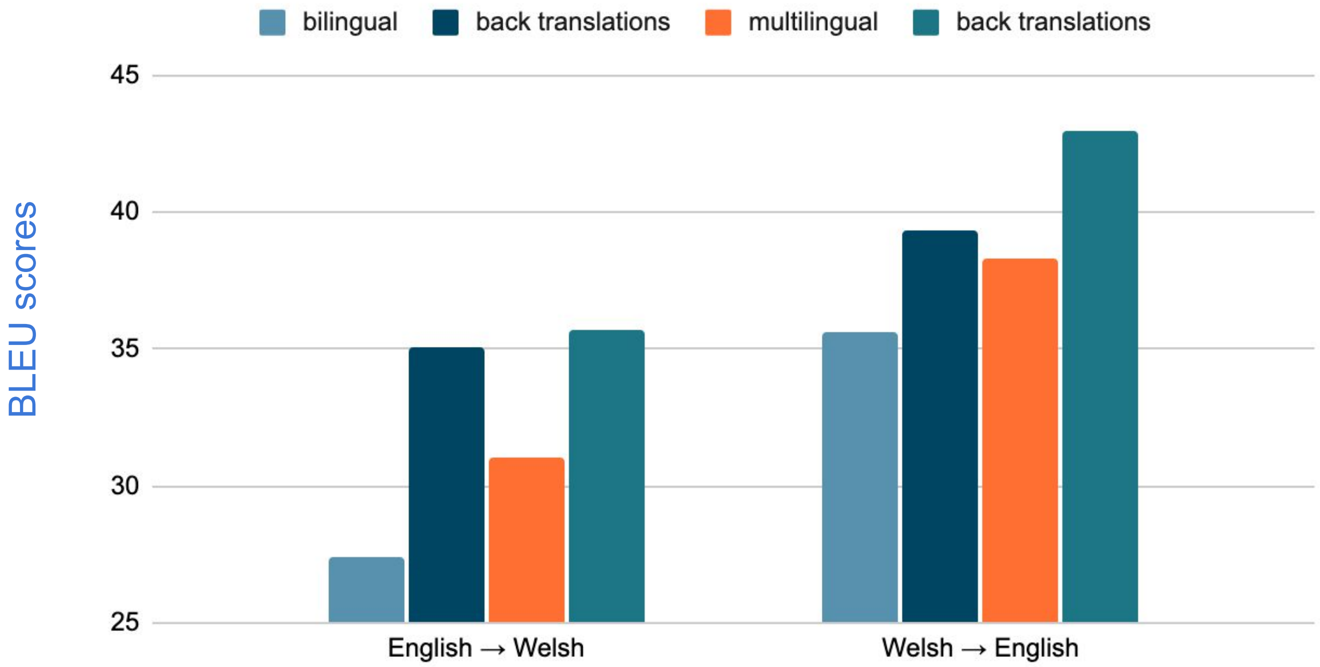


Examples of successful transfer learning

BLEU scores (in %)

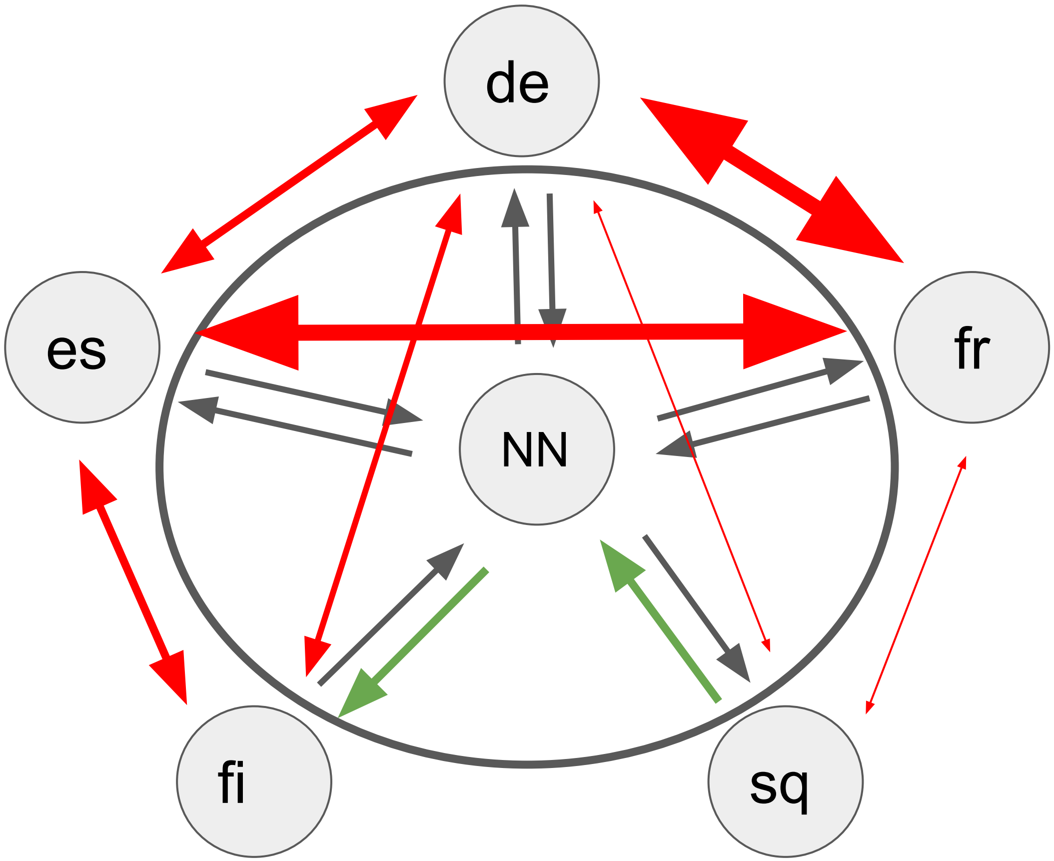
Model / test set	Belarusian → English	English → Belarusian
Belarusian - English	10.0	8.2
East Slavic languages - English	38.7	20.8
Slavic languages - English	**42.7**	**22.9**

Examples of successful transfer learning



multilingual model:
Celtic languages
and English

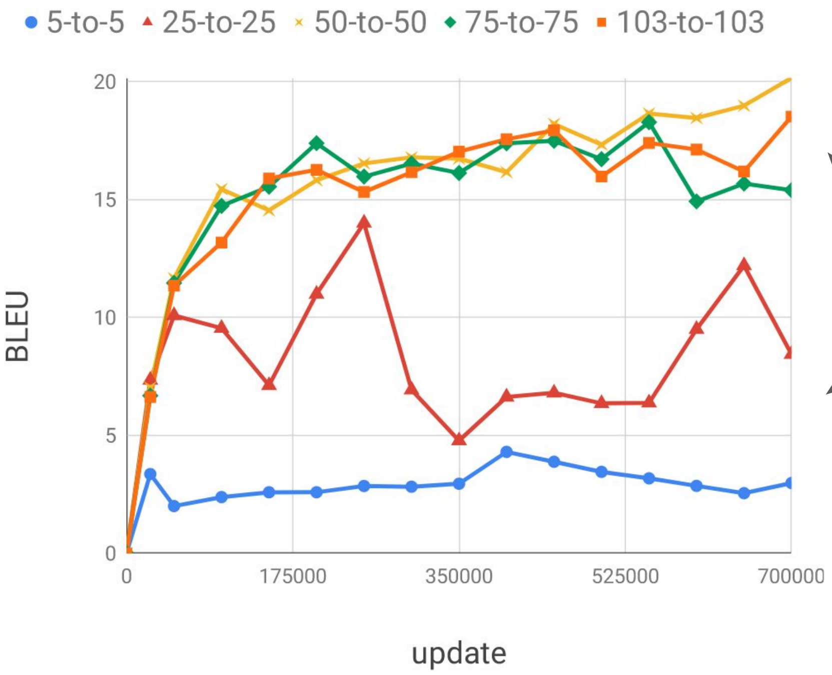
Zero shot translation directions



Zero-shot translation in massively multilingual models

Test case:

Ukrainian - Russian

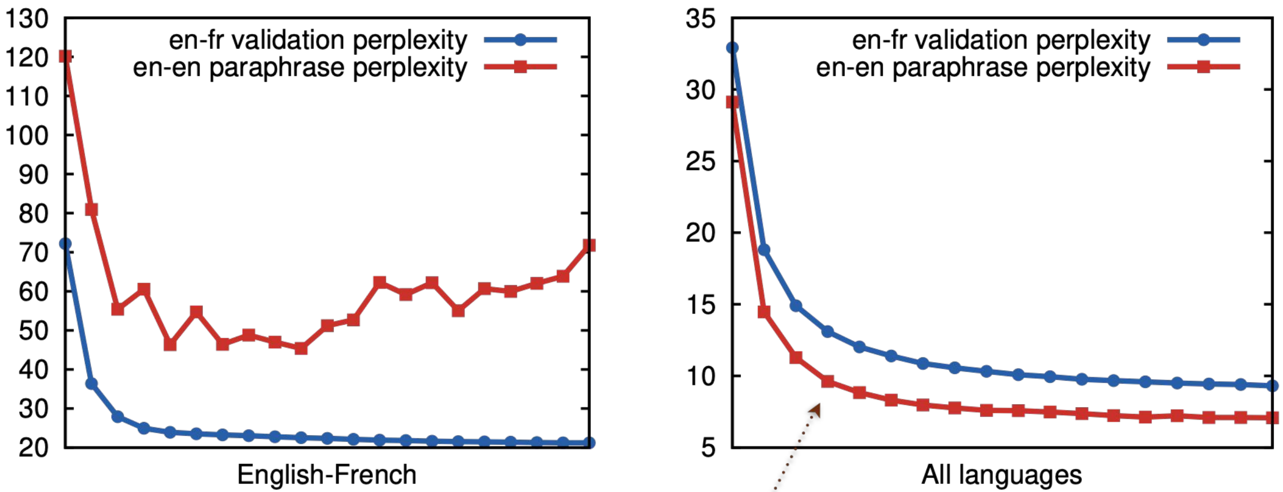


Better generalization in highly multilingual models

From: Aharoni et al. “Massively Multilingual Neural Machine Translation (NAACL, 2019)

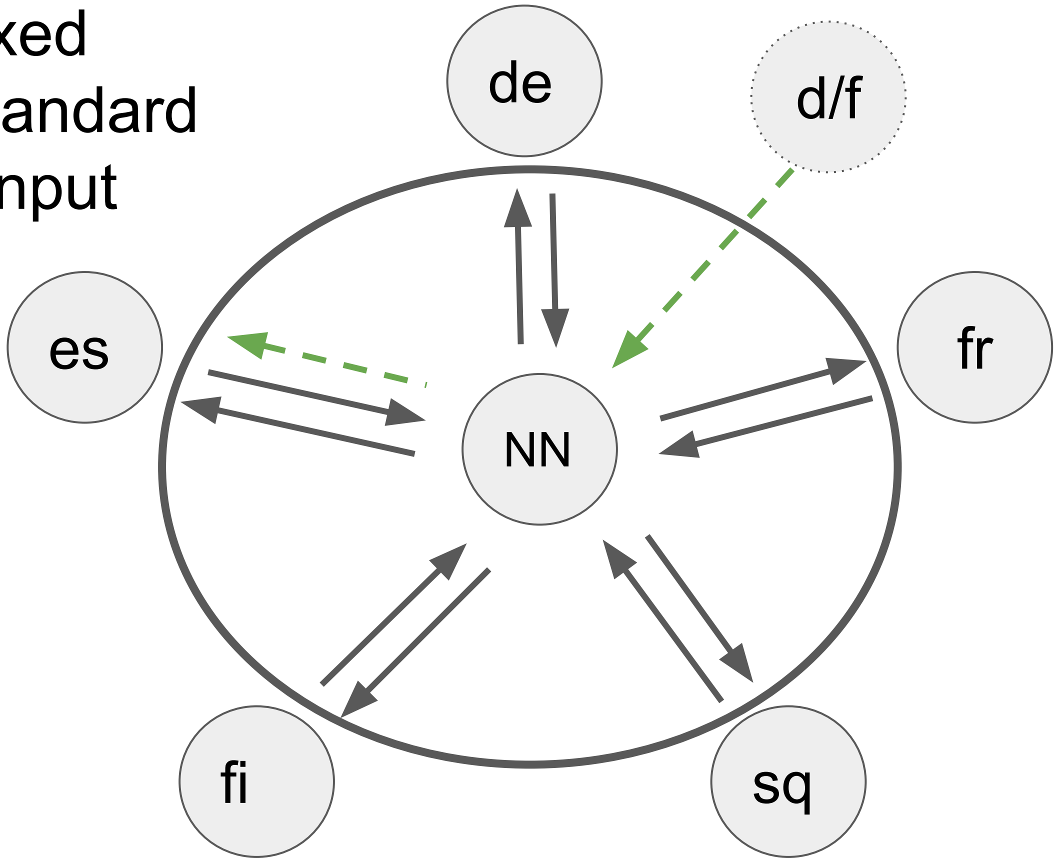
Treating paraphrasing as zero-shot translation

Learning curves during training:

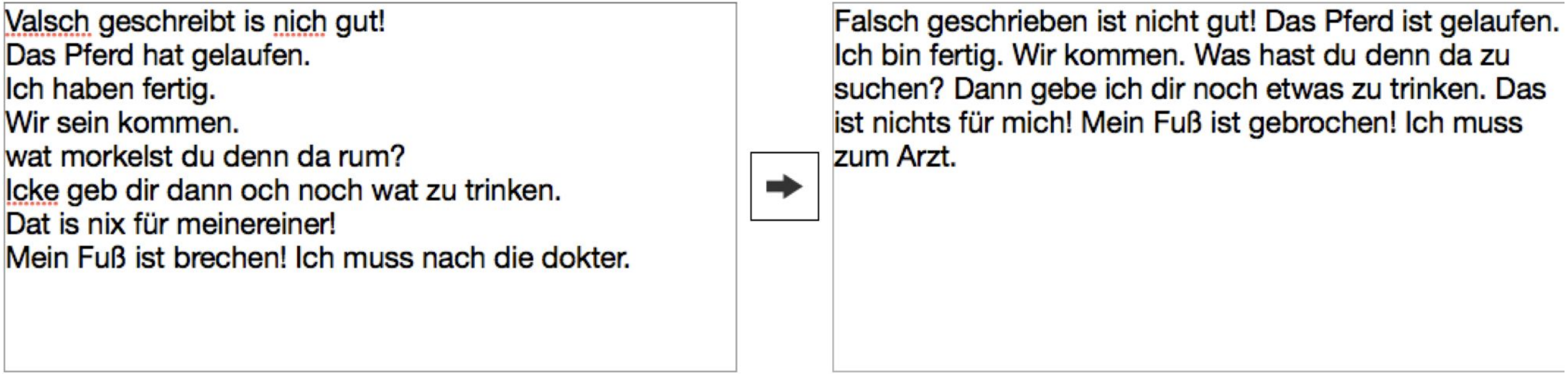


learn to recognize
paraphrased sentences

Handle mixed and non-standard language input



Multilingual NMT for text normalisation

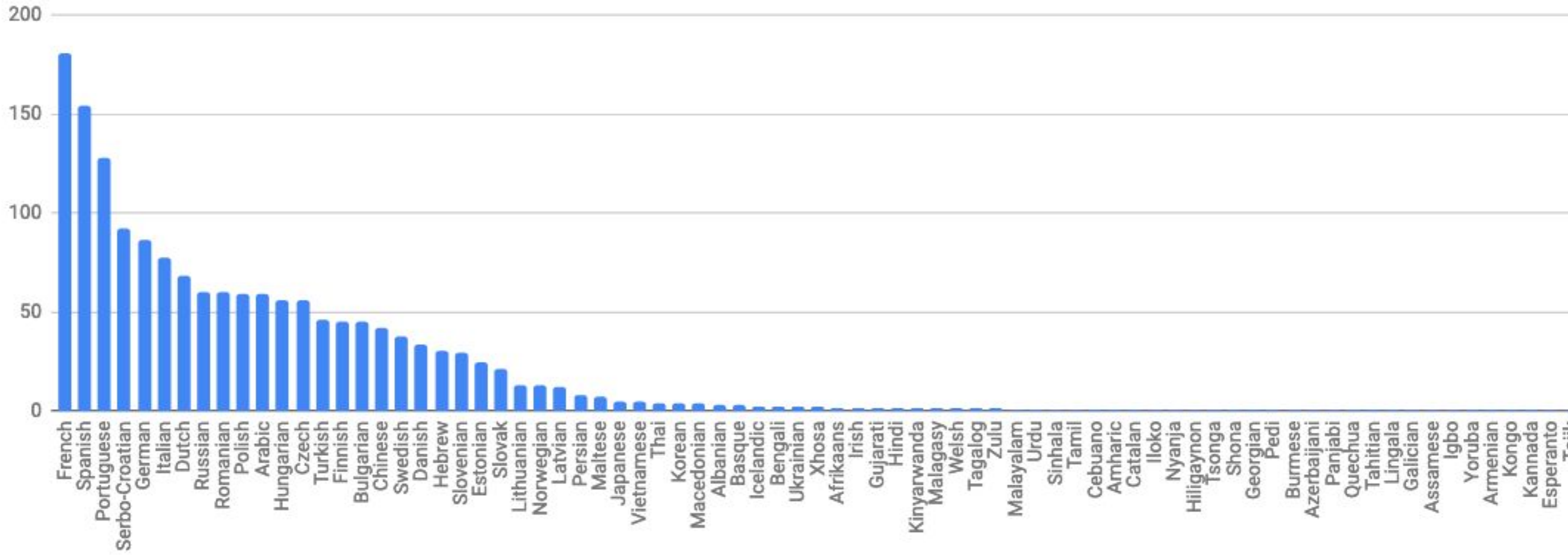


The Curse of Multilinguality

(1) Limits of training data

Size of bitexts in OPUS

(slightly outdated counts)



Languages aligned to English

Very skewed towards English-centric data and tasks

language	files	tokens	sentences	bg	ca	cs	da	de	el	en	es	et	eu	fi	fr	ga	gl	hr	hu	is	it	km									
bg	239	240.4M	12.9M							11.9M																					
ca	1,071	1.0G	54.9M								53.5M																				
cs	1,004	738.1M	52.4M							50.2M																					
da	839	690.2M	43.4M							41.9M																					
de	5,242	4.3G	274.9M							261.1M																					
el	692	592.1M	35.7M							34.6M																					
en	33,020	28.6G	1.7G	11.9M		50.2M	41.9M	261.1M	34.6M		396.5M	8.6M		15.3M	266.8M	2.0M		11.1M	12.7M	5.7M	120.1M	65.1k	4.0M	8.0M	8.2M	1.6M	31.4k	92.1k	98.5M	59.1M	4
es	9,300	8.2G	479.7M		53.5M					396.5M			2.4M				12.4M														
et	172	130.5M	8.9M							8.6M																					
eu	49	36.2M	2.5M								2.4M																				
fi	307	226.2M	16.1M							15.3M																					
fr	5,391	5.9G	280.7M							266.9M																				2.7M	
ga	40	48.7M	2.1M							2.0M																					
gl	249	152.1M	12.6M								12.4M																				
hr	222	175.9M	11.5M							11.1M																					
hu	254	208.2M	13.4M							12.7M																					
is	115	86.4M	6.0M							5.7M																					
it	2,403	2.3G	124.2M							120.1M																					
km	2	2.0M	68.2k							65.1k																					
ko	81	58.3M	4.1M							4.0M																					
lt	161	129.2M	8.4M							8.0M																					
lv	164	140.9M	8.6M							8.2M																					
mt	33	33.0M	1.7M							1.6M																					
my	1	0.9M	33.9k							31.4k																					
ne	2	4.0M	94.7k							92.1k																					
nl	2,024	1.7G	105.4M							98.5M					2.7M																
no	1,182	867.6M	61.5M							59.1M																					
pl	927	738.9M	48.1M					0.9M		45.4M																					
ps	1	0.9M	27.4k							26.3k																					
pt	2,053	1.9G	106.1M							102.6M																					
ro	268	269.0M	14.4M							13.4M																					
ru	108	98.4M	6.1M							5.4M																					
si	5	7.6M	0.2M							0.2M																					
sk	261	210.8M	13.5M							13.0M																					
sl	151	141.9M	7.8M							7.5M																					
so	1	0.5M	20.5k							14.9k																					
sv	882	702.3M	46.0M							44.1M																					
sw	3	3.8M	0.2M							0.1M																					
tl	5	7.8M	0.3M							0.2M																					

ParaCrawl

WMT news translation tasks

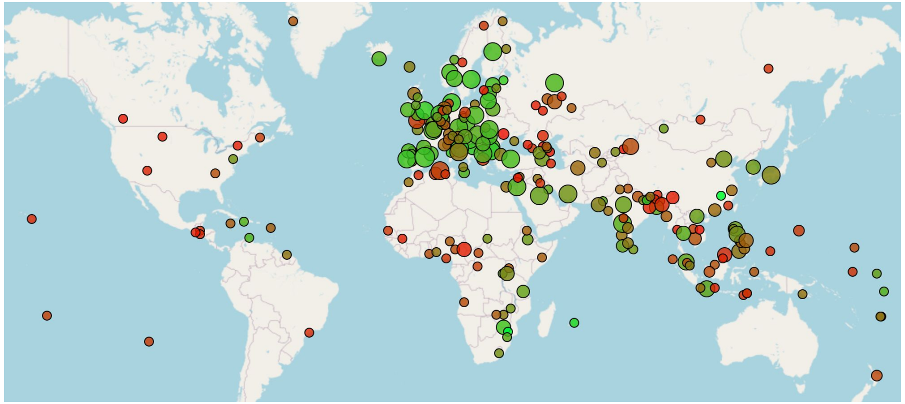
Both directions

- Chinese to/from English
- German to/from English: document-level
- Hebrew to/from English: low-resource
- Japanese to/from English
- Russian to/from English
- Ukrainian to/from English

Single direction

- Czech to Ukrainian: non-English
- English to Czech

OPUS-100
MASSIVELY MULTILINGUAL TRANSLATION



https://github.com/Helsinki-NLP/Tatoeba-Challenge

OPUS

Tatoeba Translation Challenge

Realistic MT data sets with large language coverage (currently: 557 languages)

- No artificial low-resource scenarios
- Straightforward to use (train/dev/test splits)
- Consistent language labels + writing script information

Benchmarks

- Tatoeba collection of user-contributed translations
- Continuously updated

(2) Limits of generalisations & transfer learning in NMT

BLEU scores (in %)

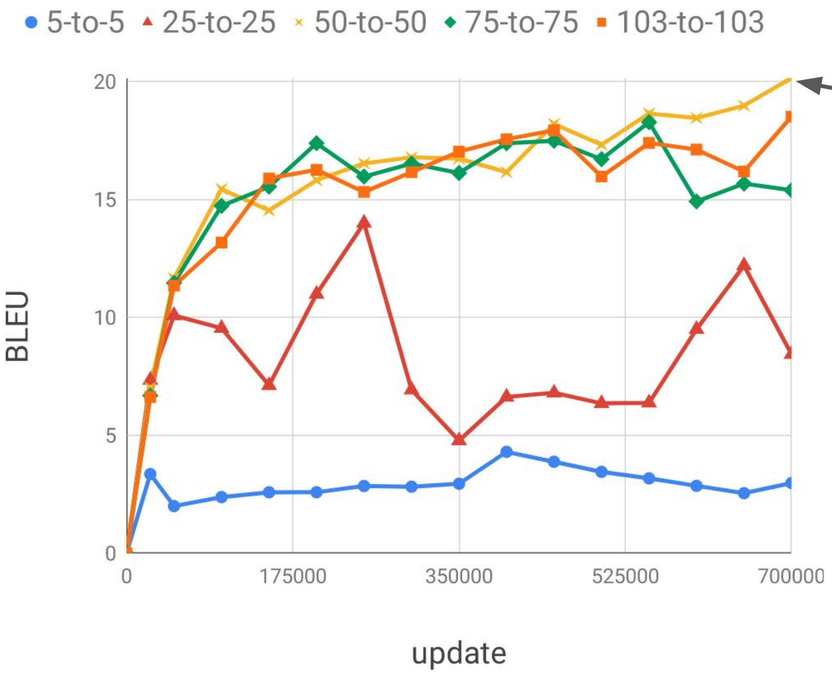
Model / test set	Belarusian → English	English → Belarusian
Belarusian - English	10.0	8.2
East Slavic languages - English	38.7	20.8
Slavic languages - English	**42.7**	**22.9**
Indo-European languages - English	41.7	18.1

(increasing language coverage while keeping the model size constant)

Zero-shot translation in massively multilingual models

Test case:

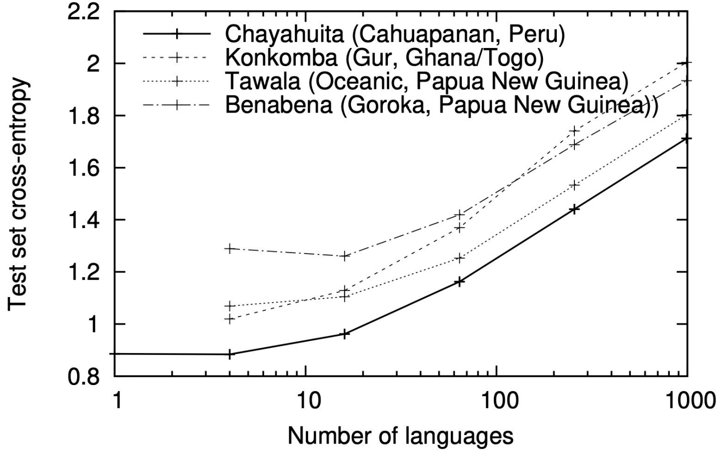
Ukrainian - Russian



Sweet spots of multlinguality?

From: Aharoni et al. “Massively Multilingual Neural Machine Translation (NAACL, 2019)

(3) Limits of the model capacity

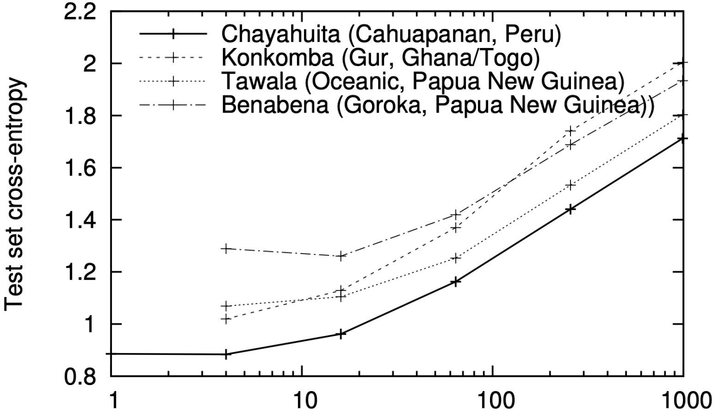


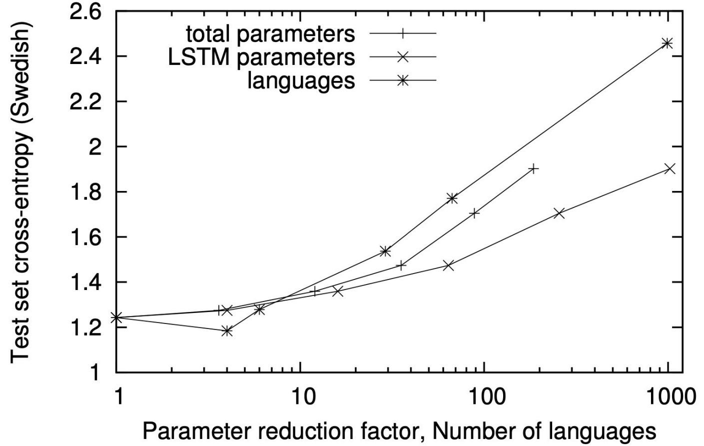
Testing the model capacity when adding more languages

(similar patterns for adding languages in random order or according to typological relationship)

(from *“Continuous multilinguality with language vectors”*)

(3) Limits of the model capacity



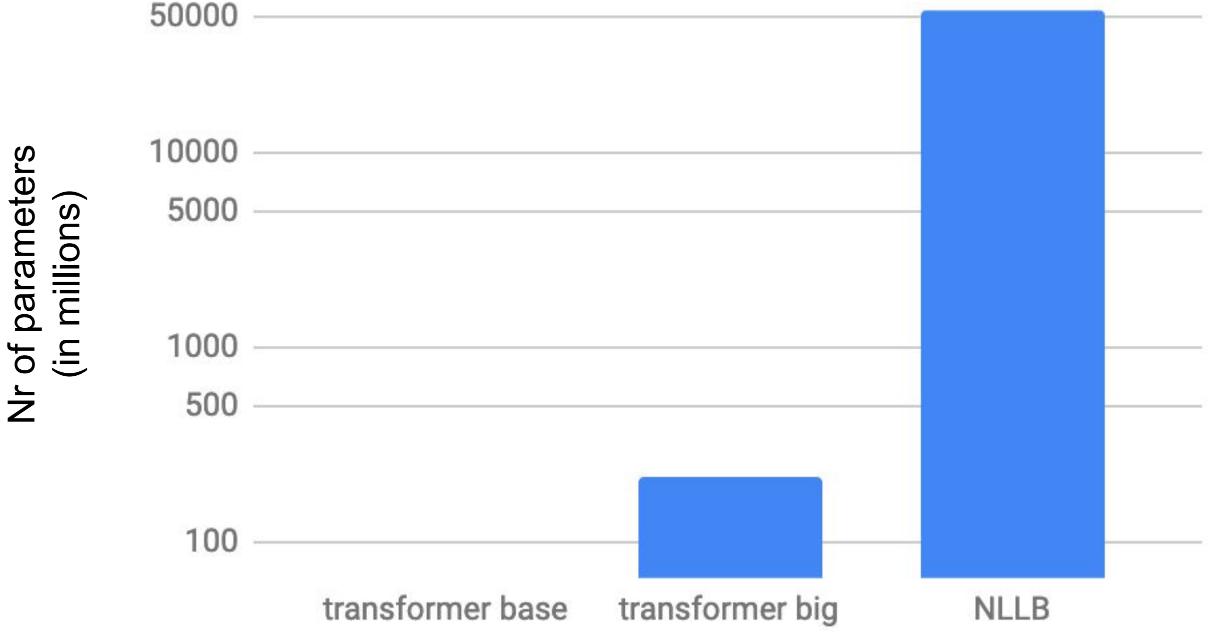


Comparing the impact of

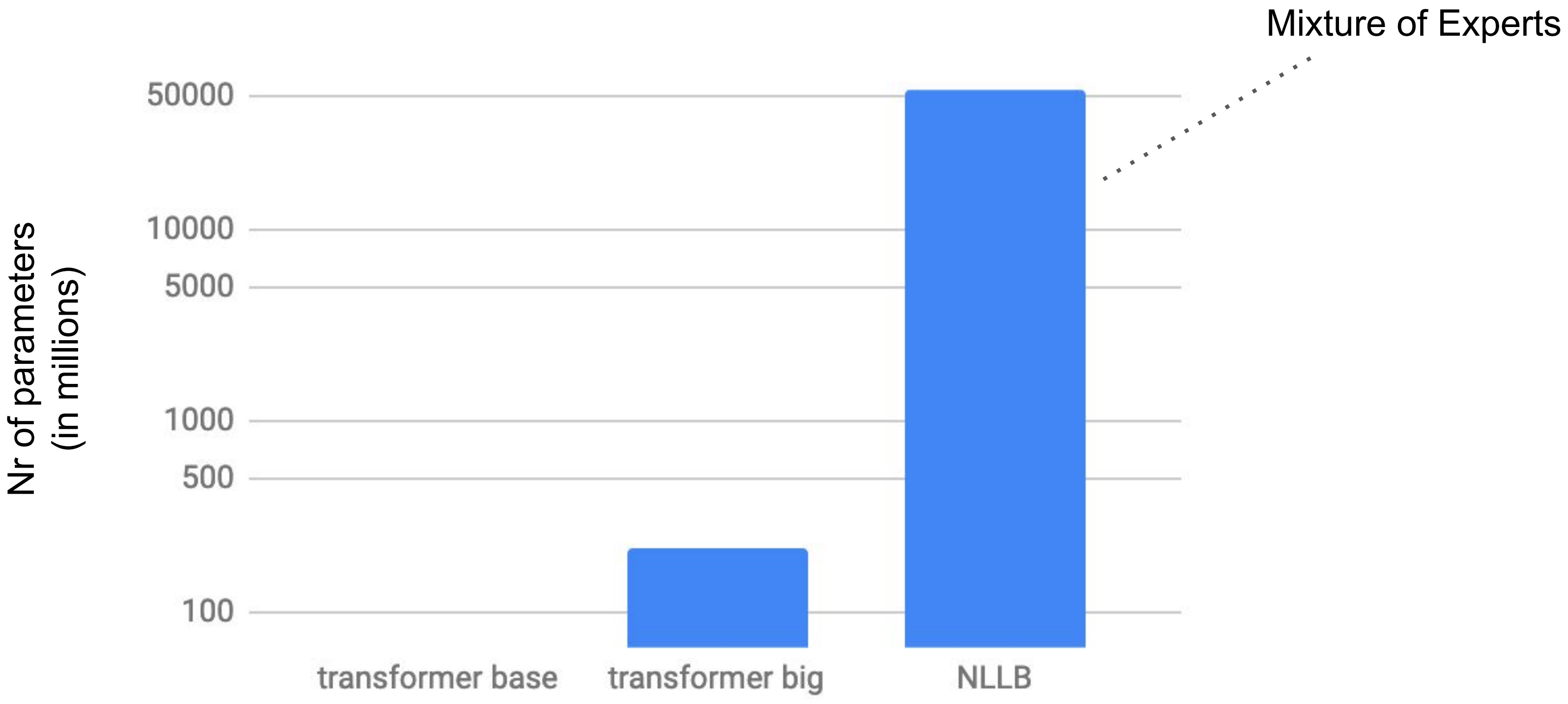
- adding languages
- decreasing model size

(from *“Continuous multilinguality with language vectors”*)

Growing model size for multilingual models

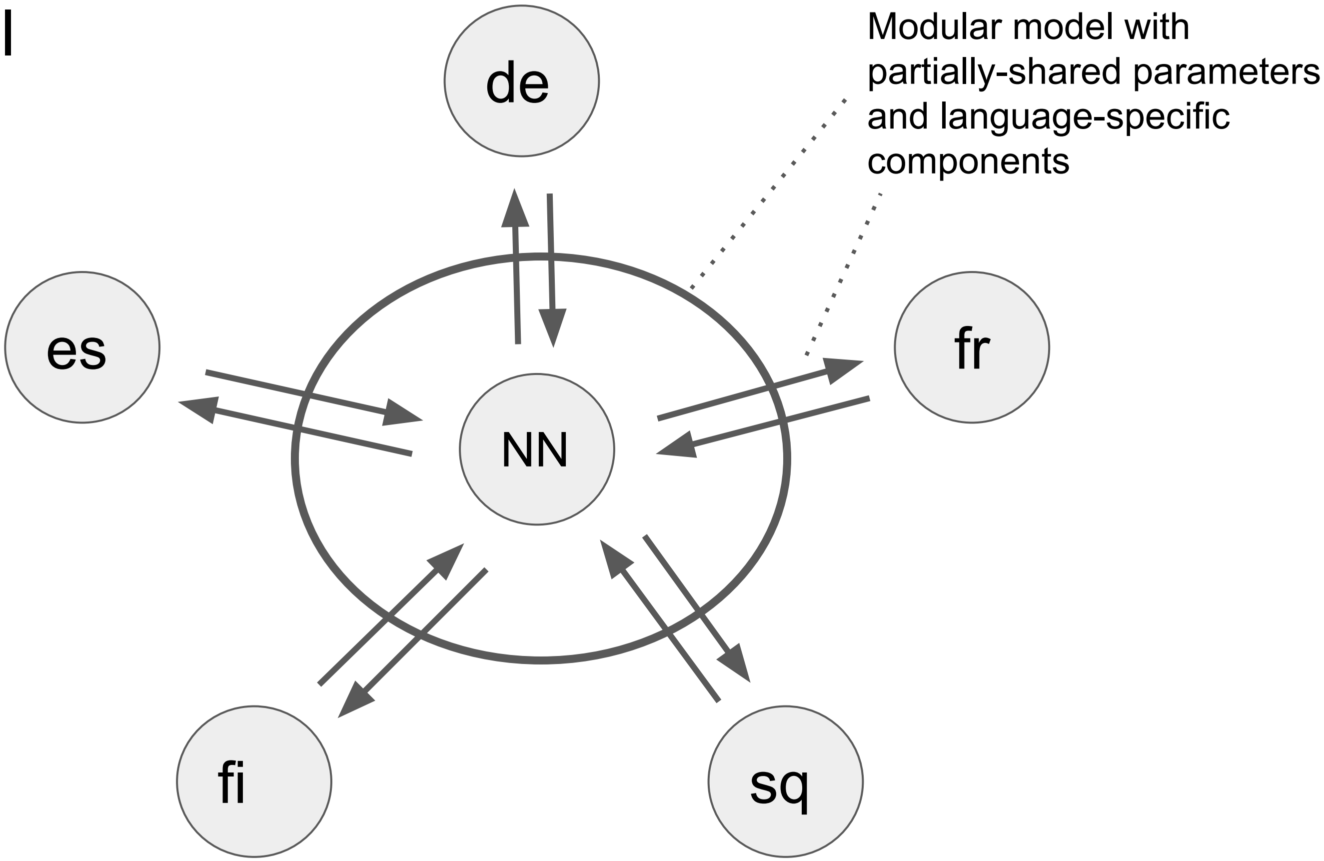


Growing model size for multilingual models

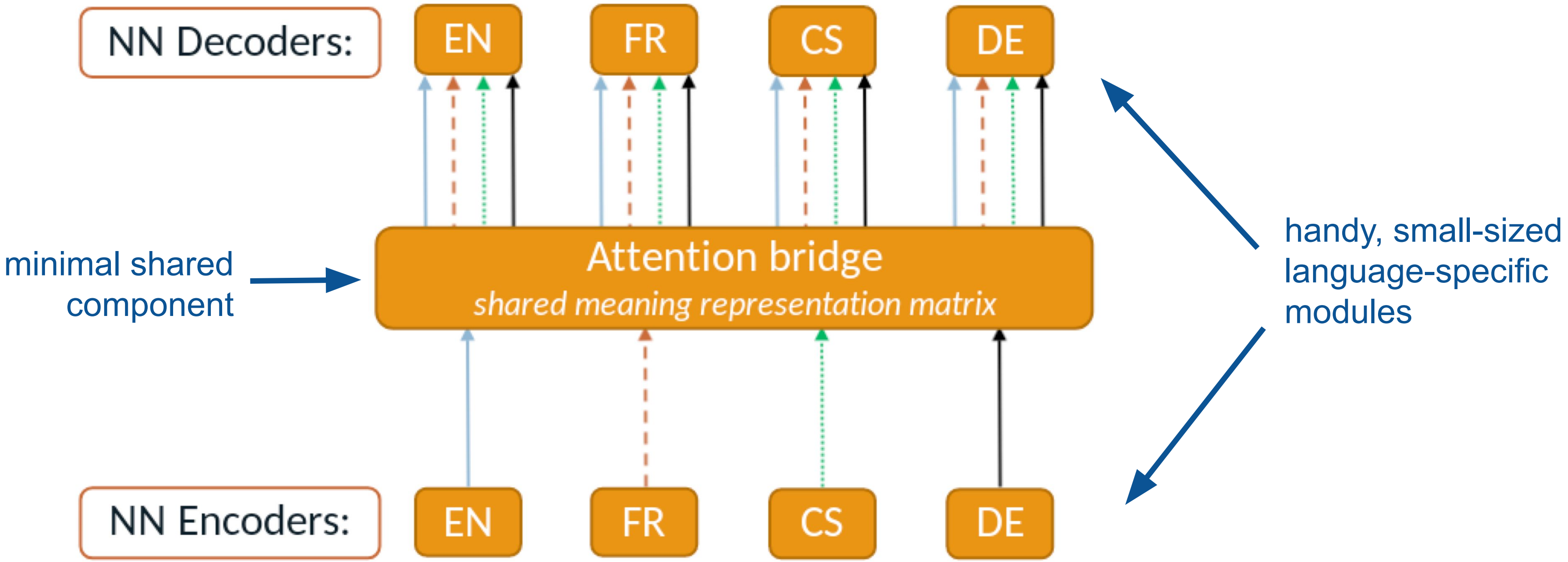


Back to Modularity

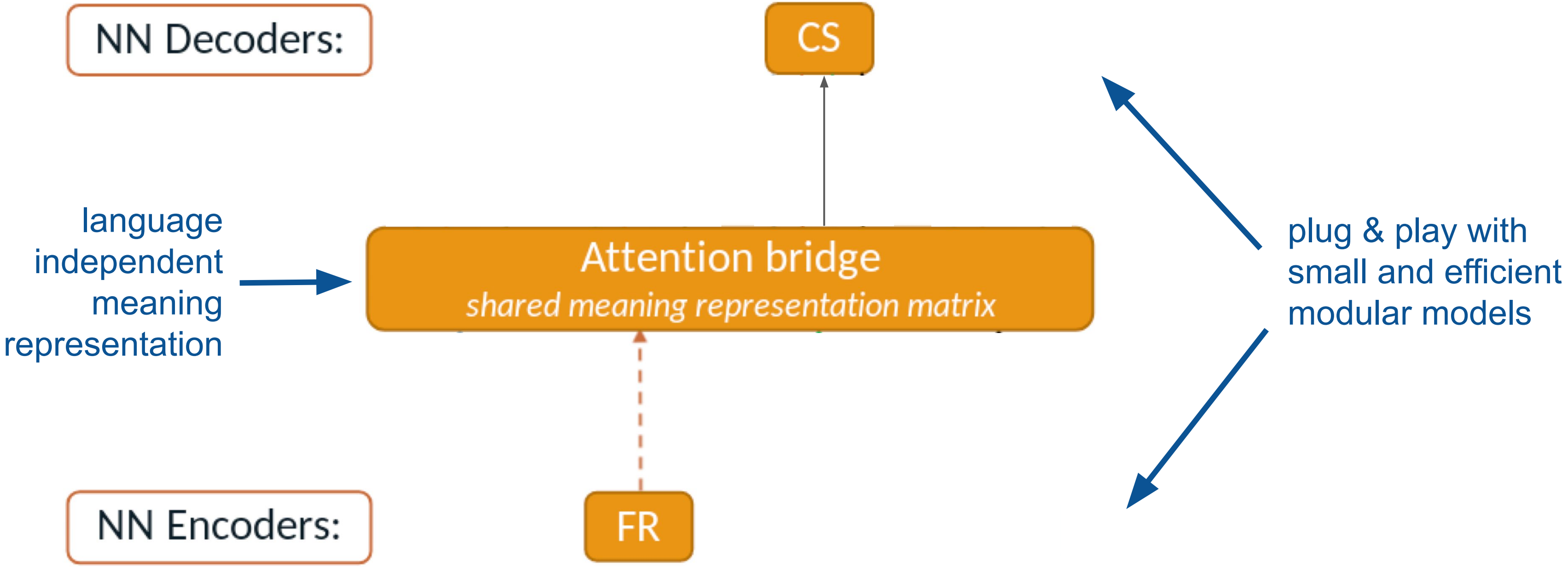
Multilingual NMT



At the MT marathon 5 years ago …



At the MT marathon 5 years ago …



What happened since then?

Modular NLP is increasingly popular

- Partial sharing and task-specific components
- Adapters and hyper networks
- Gated routing and mixture of experts (see NLLB and GPT-4)

Many open questions

- What to share and how much?
- Hierarchical models and language clusters?
- Efficient training with optimal routing and communication

What happened s... ?

Modular NLP is increa...

- Partial sharing a...
- Adapters and hy...
- Gated routing a... T-4)

Many open que... tio...

- What to sha...
- Hierarchical m... s an... ag...
- Efficient training ... ith op... out ... hur...

What we want from a scalable mNMT system

Must-have Feats of a Scalable mNMT System:

- Allow for versatile parameter sharing
- Efficient GPU allocation
- Supports addition of new language pairs
- Provide tools for mNMT data management
- Efficient inference

What to do?

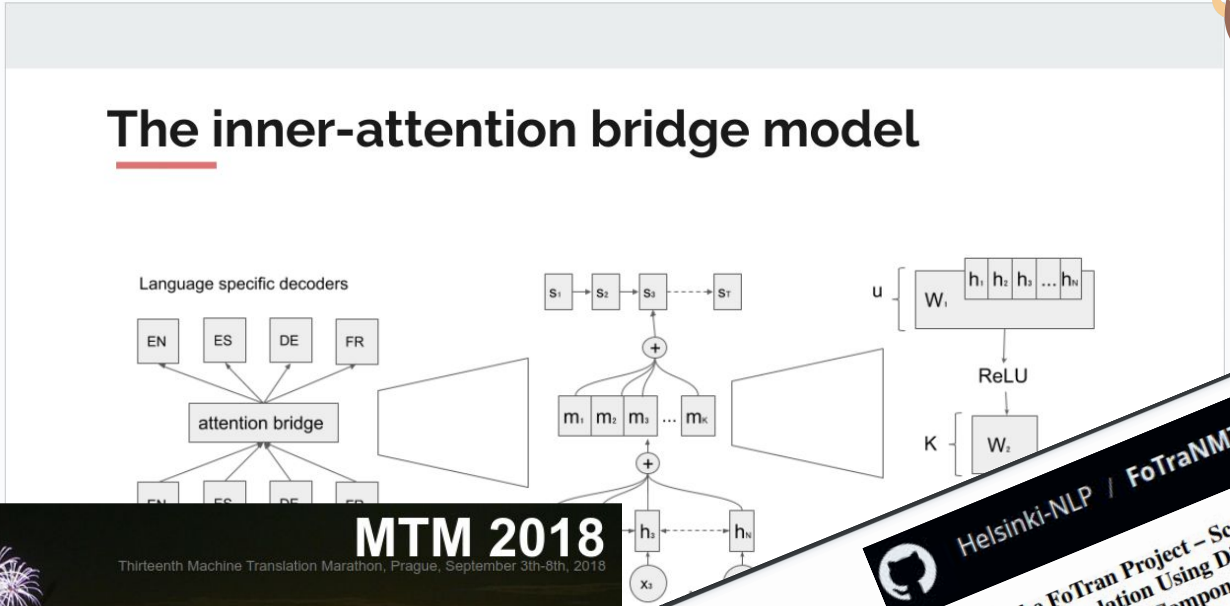
Use our toolkit(*,**) to train modular systems:

MAMMOTH: MAssively Multilingual Modular Open Translation @ Helsinki

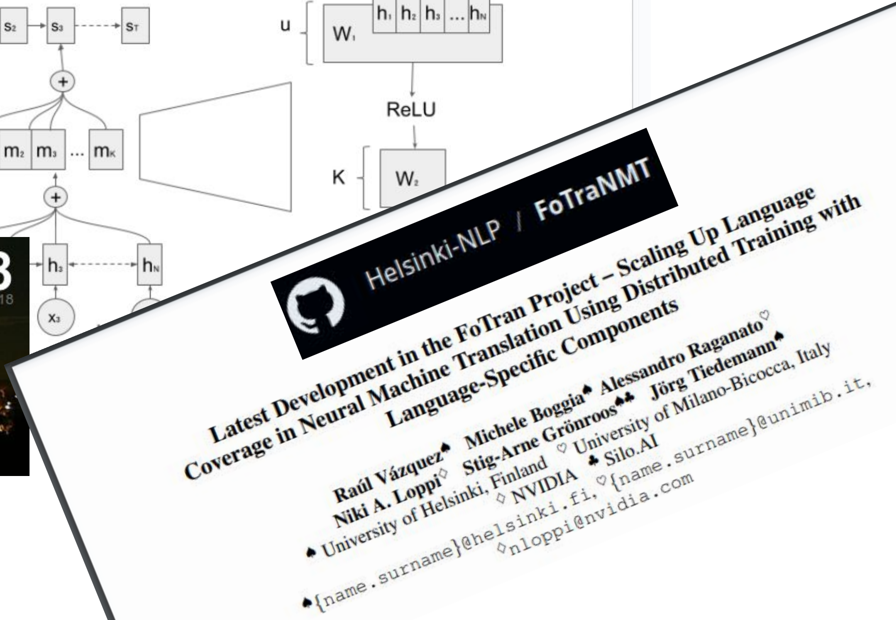
https://github.com/Helsinki-NLP/mammoth

** still a beta version*
*** we built on top of OpenNMT-py but the code has changed so much that it cannot be named after the parent codebase*

The lore


The inner-attention bridge model
Language specific decoders
EN
ES
DE
FR
attention bridge
u
W₁
h₁ h₂ h₃ ... h_N
ReLU
K
W₂
s₁ s₂ s₃ s_T
m₁ m₂ m₃ ... m_K
h₃ h_N
x₃


MTM 2018
Thirteenth Machine Translation Marathon, Prague, September 3th-8th, 2018


Helsinki-NLP / FoTraNMT
Latest Development in the FoTran Project – Scaling Up Language Coverage in Neural Machine Translation Using Distributed Training with Language-Specific Components
Raúl Vázquez♠ Michele Boggia♠ Alessandro Raganato♡
Niki A. Loppi◊ Stig-Arne Grönroos♠♣ Jörg Tiedemann♠
♠ University of Helsinki, Finland ♡ University of Milano-Bicocca, Italy
◊ NVIDIA ♣ Silo.AI
♠{name.surname}@helsinki.fi, ♡{name.surname}@unimib.it,
◊nloppi@nvidia.com

Scalable mNMT Systems

Features	The ABCD of mNMT
• Parameter sharing	A) Anatomy of parameter sharing B) Bridges and structures for sharing
• GPU allocation & communication • Add new languages	C) Communication chaos
• mNMT data	D) Dearth of data
• Efficient inference	

Scalable mNMT Systems

We showcase mNMT using MAMMOTH

Features	The ABCD of
• Parameter sharing	A) Anatomy o
	B) Bridges and structures for sharing
• GPU allocation & communication	C) Communication chaos
• Add new languages	
• mNMT data	D) Dearth of data
• Efficient inference	

A) Anatomy of parameter sharing

- Parameter sharing is tied to transfer learning
- The trichotomy of this choice:
  - full sharing,
  - no sharing, and
  - everything in between

A) Anatomy of parameter sharing

Fully shared (Johnson et al., 2017)

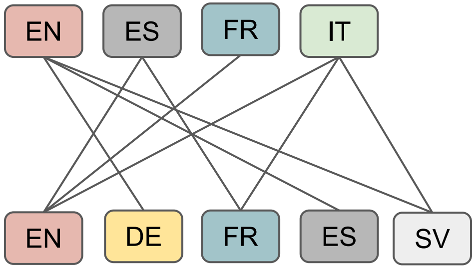
- Simple and effective
- Use of language tags (prompt-based learning predecessor; Liu et al.,2023)

```
<2tgt> The rise of the radical right across Europe is a
       symptom of a failing capitalism.
```

A) Anatomy of parameter sharing

No shared parameters (Escolano et al., 2021)

- Exploits the encoder-decoder NMT architecture
- Increases the data (& its distribution) used to train encoder/decoder modules
- Easy to add modules (no need to re-train)



A) Anatomy of parameter sharing

Partial sharing schemes (or everything in between)

- Myriad of research works
- Roughly, we classify into:

Transversal

Longitudinal

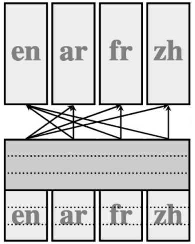
Embeddings
or vocab hacks

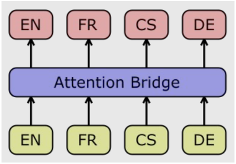
A) Anatomy of parameter sharing

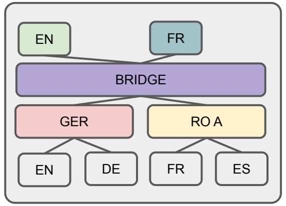
Partial sharing schemes (or everything in between)

- Myriad of research works
- Roughly, we classify into:

Transversal







Longitudinal

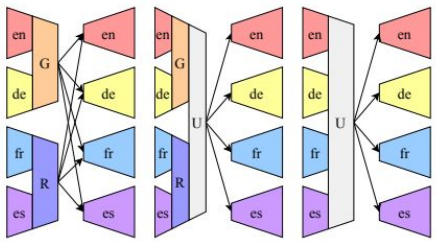
Embeddings
or vocab hacks

A) Anatomy of parameter sharing

Partial sharing schemes (or everything in between)

- Myriad of research works
- Roughly, we classify into:

Transversal
(e.g.,Purason& Tättar, 2022)



(a) Language group sharing **(b)** Language group and universal sharing (tiered) **(c)** Universal sharing (between all languages)

Longitudinal

Embeddings
or vocab hacks

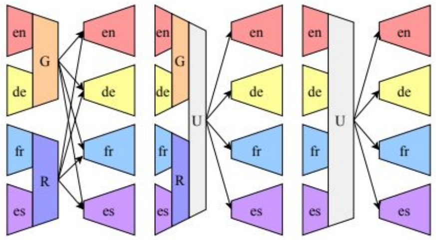
A) Anatomy of parameter sharing

Partial sharing schemes (or everything in between)

- Myriad of research works
- Roughly, we classify into:

Transversal

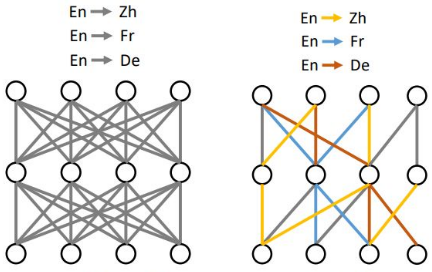
(e.g.,Purason& Tättar, 2022)



(a) Language group sharing **(b)** Language group and universal sharing (tiered) **(c)** Universal sharing (between all languages)

Longitudinal

(e.g.,LaSS Lin et al., 2021)



(a) Full network (b) LaSS

Embeddings

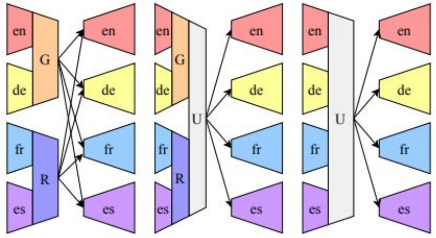
or vocab hacks

A) Anatomy of parameter sharing

Partial sharing schemes (or “everything in between”)

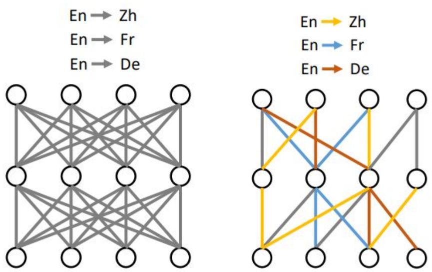
- Myriad of research works
- Roughly, we classify into:

Transversal
(e.g.,Purason& Tättar, 2022)



(a) Language group sharing (b) Language group and universal sharing (tiered) (c) Universal sharing (between all languages)

Longitudinal
(e.g., Lin et al., 2021)



(a) Full network (b) LaSS

Embeddings
or vocab hacks

Share all vocabs:
- naively (Johnson et al, 2017)
- Temp.-based sampling (Aharoni, 2018)

Dynamic vocab adaptation (Lakew, 2018)
Vocab substitution in incremental training (Chronopoulou, 2020; Garcia, 2021; Huang, 2022)

A) Anatomy of parameter sharing

We break down mNMT training into a series of smaller “tasks”

- A task requires specific modules
- A task is done on a specific device
- A task corresponds to a specific (parallel) corpus

```
class TaskSpecs():
    node_rank: int
    local_rank: int
    src_lang: str
    tgt_lang: str
    encoder_id: List[str]
    decoder_id: List[str]
    corpus_id: str
    weight: int
    corpus_opt: dict
```

A centralized manager handles tasks synchronization

A) Anatomy of parameter sharing

- **full sharing**

```yaml
data:
  train_ar-ar:
    src_tgt: ar-ar
    enc_sharing_group: ["all"]
    dec_sharing_group: ["all"]
    path_src: /path/to/train.ar
    path_tgt: /path/to/train.ar
    transforms: [filtertoolong, bart]
  train_ar-en:
    src_tgt: ar-en
    enc_sharing_group: ["all"]
    dec_sharing_group: ["all"]
    path_src: /path/to/train.ar
    path_tgt: /path/to/train.en
    transforms: [filtertoolong]
  train_en-ar:
    src_tgt: en-ar
    enc_sharing_group: ["all"]
    dec_sharing_group: ["all"]
    path_src: /path/to/train.en
    path_tgt: /path/to/train.ar
    transforms: [filtertoolong]
  train_en-en:
    src_tgt: en-en
    enc_sharing_group: ["all"]
    dec_sharing_group: ["all"]
    path_src: /path/to/train.en
    path_tgt: /path/to/train.en
    transforms: [filtertoolong, bart]
```

A) Anatomy of parameter sharing

- full sharing
- **no sharing**

```yaml
data:
  train_ar-ar:
    src_tgt: ar-ar
    enc_sharing_group: ["ar"]
    dec_sharing_group: ["ar"]
    path_src: /path/to/train.ar
    path_tgt: /path/to/train.ar
    transforms: [filtertoolong, bart]
  train_ar-en:
    src_tgt: ar-en
    enc_sharing_group: ["ar"]
    dec_sharing_group: ["en"]
    path_src: /path/to/train.ar
    path_tgt: /path/to/train.en
    transforms: [filtertoolong]
  train_en-ar:
    src_tgt: en-ar
    enc_sharing_group: ["en"]
    dec_sharing_group: ["ar"]
    path_src: /path/to/train.en
    path_tgt: /path/to/train.ar
    transforms: [filtertoolong]
  train_en-en:
    src_tgt: en-en
    enc_sharing_group: ["en"]
    dec_sharing_group: ["en"]
    path_src: /path/to/train.en
    path_tgt: /path/to/train.en
    transforms: [filtertoolong, bart]
```

A) Anatomy of parameter sharing

- full sharing
- no sharing
- and **everything in between**

```yaml
data:
  train_ar-ar:
    src_tgt: ar-ar
    enc_sharing_group: ["ar", "all"]
    dec_sharing_group: ["ar1", "all", "ar2"]
    path_src: /path/to/train.ar
    path_tgt: /path/to/train.ar
    transforms: [filtertoolong, bart]
  train_ar-en:
    src_tgt: ar-en
    enc_sharing_group: ["ar", "all"]
    dec_sharing_group: ["en1", "all", "en2"]
    path_src: /path/to/train.ar
    path_tgt: /path/to/train.en
    transforms: [filtertoolong]
  train_en-ar:
    src_tgt: en-ar
    enc_sharing_group: ["en", "all"]
    dec_sharing_group: ["ar1", "all", "ar2"]
    path_src: /path/to/train.en
    path_tgt: /path/to/train.ar
    transforms: [filtertoolong]
  train_en-en:
    src_tgt: en-en
    enc_sharing_group: ["en", "all"]
    dec_sharing_group: ["en1", "all", "en2"]
    path_src: /path/to/train.en
    path_tgt: /path/to/train.en
    transforms: [filtertoolong, bart]
```

A) Anatomy of parameter sharing

This relies on an implementation of *coders as stacks of layer stacks

```
class LayerStackEncoder(EncoderBase):
    def __init__(self, embeddings, encoders):
        super().__init__()

        self.embeddings = embeddings
        self.encoders: nn.ModuleList[nn.ModuleDict] = encoders

class LayerStackDecoder(DecoderBase):
    def __init__(self, embeddings, decoders):
        super().__init__()

        self.embeddings = embeddings
        self.decoders: nn.ModuleList[nn.ModuleDict] = decoders
```

```
def forward(self, src, lengths=None, **kwargs):
    # wrapper embeds src and creates mask
    emb = self.embeddings(src)
    emb = emb.transpose(0, 1).contiguous()
    mask = ~sequence_mask(lengths).unsqueeze(1)

    output = emb
    for active_id, stacks in zip(self._active, self.encoders):
        encoder = stacks[active_id]
        # Throw away emb, lengths, mask
        _, output, _, _ = encoder.forward(
            output,
            lengths=lengths,
            skip_embedding=True,
            mask=mask,
        )

    # only at the end transpose back to timestep-first
    output = output.transpose(0, 1).contiguous()
    return emb, output, lengths, mask
```

A) Anatomy of parameter sharing

We can define stacks with arbitrary number of layers

```yaml
rnn_size: 512
word_vec_size: 512
transformer_ff: 2048
heads: 8
enc_layers: [5, 1]
dropout: 0.1
```

```python
stacks[module_id] = AdaptedTransformerEncoder(
    n_layers,
    model_opt.enc_rnn_size,
    model_opt.heads,
    model_opt.transformer_ff,
    model_opt.dropout[0] if type(model_opt.dropout) is list else model_opt.dropout,
    (
        model_opt.attention_dropout[0]
        if type(model_opt.attention_dropout) is list
        else model_opt.attention_dropout
    ),
    None,  # embeddings,
    model_opt.max_relative_positions,
    pos_ffn_activation_fn=model_opt.pos_ffn_activation_fn,
)
```

A) Anatomy of parameter sharing

Our research:

Effects of **Sharing more parameters VS Adding lang. pairs** on (Boggia et al.,2023):

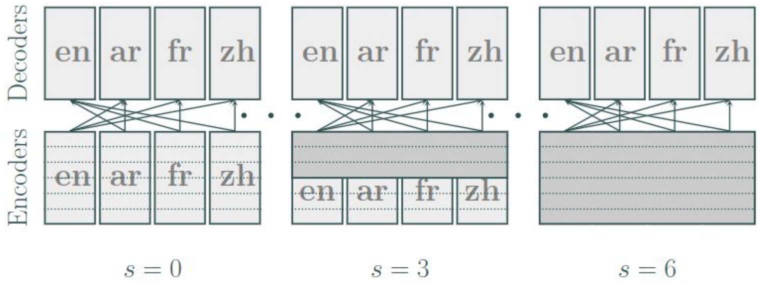
- Task fitness OR *how adequate they are for machine translation?*
- Lang. independence OR *how independent of the source language they are?*
- Semantic content OR *what semantic information they convey?*

Dozens of Translation Directions or Millions of Shared Parameters? Comparing Two Types of Multilinguality in Modular Machine Translation

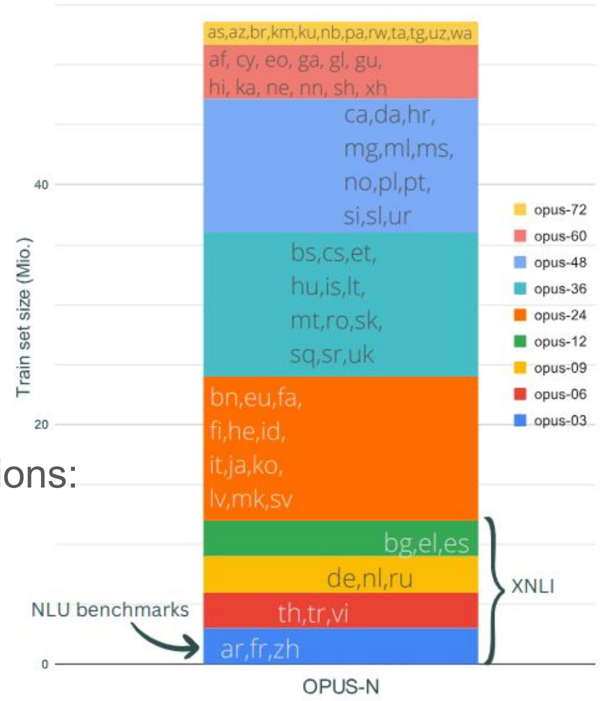
Michele Boggia♠ Stig-Arne Grönroos♠ Niki Andreas Loppi♢ Timothee Mickus♠ Alessandro Raganato♡ Jörg Tiedemann♠ Raúl Vázquez♠

A) Anatomy of parameter sharing

Sharing more parameters



Adding lang. pairs



∀ language *L* in a dataset, we consider 3 translation directions:

1. *L*-to-English,
2. English-to-*L*, and
3. *L*-to-*L* denoising auto-enc.

A) Anatomy of parameter sharing

Conclusions **Sharing more parameters VS Adding lang. pairs** (Boggia et al.,2023)

- MAMMOTH made this study possible

Broadly: Sharing parameters and multiplying languages affect differently multilingual NLP systems.

Setting right the number of shared parameters brings higher performances and more reliable representations, but the optimal number of shard layers depends on the task.

- Go read the paper for more ;)

A) Anatomy of parameter sharing

Pikku SPOILER

DO: Spend effort on tuning the level of parameter sharing for the task at hand.

DON'T: Spend too much effort on acquiring data for additional languages

Conclusions **Sharing more parameters VS Adding lang**

- MAMMOTH made this study possible

Broadly: Sharing parameters and multiplying languages a

NLP systems.

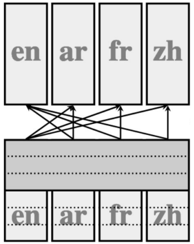
Setting right the number of shared parameters brings higher performances and more reliable representations, but the optimal number of shard layers depends on the task.

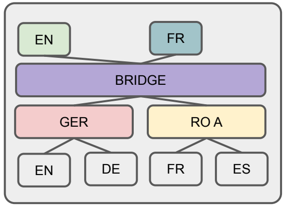
- Go read the paper for more ;)

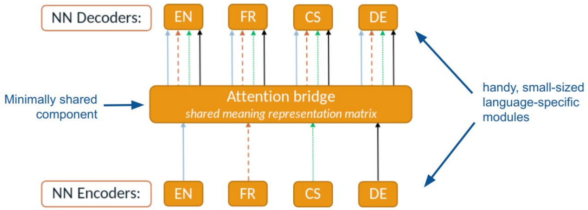
B) Bridges and structures for sharing

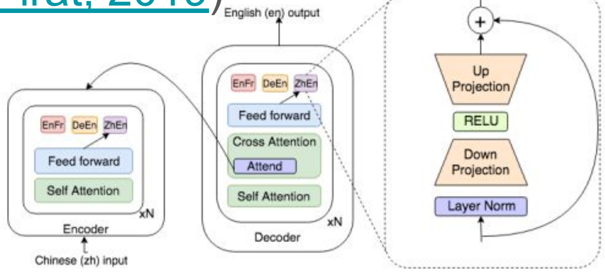
What structure we want to explore for the shared parameters???

- fully shared layers
  - Transformer-based
  - FFWD
  - Attention bridge (remember the lore?)
  - RNN, CNN?
- Adapters ([Bapna & Firat, 2019](#))



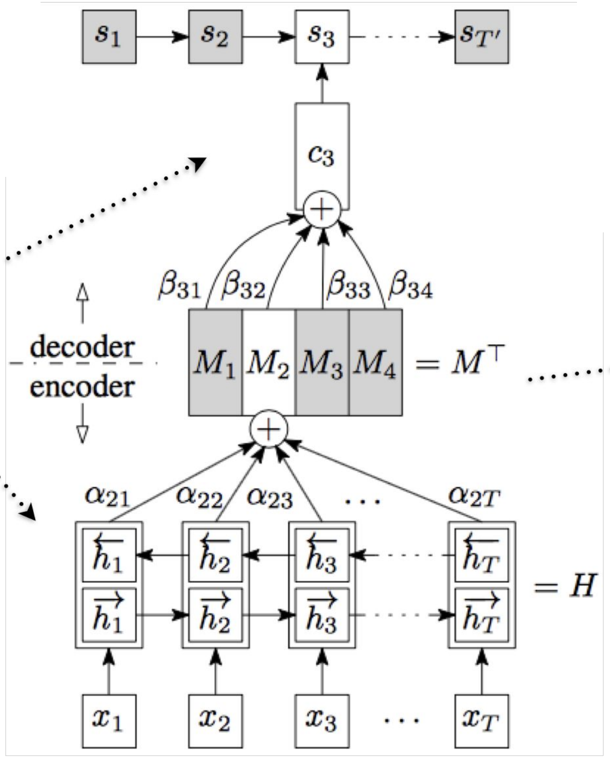






The attention bridge model

language-specific parameters



shared among all language pairs

Rotate languages in scheduled training!

Architecture proposed by Cífka and Bojar (2018). *Our implementation in OpenNMT-py (MTM2018)*

B) Bridges and structures for sharing

We saw stacks, we also have:

- **bridges**, shared across all tasks

Multiple architectures

```
class PerceiverAttentionBridgeLayer(BaseAttentionBridgeLayer):
        return alphas, self_attention_output
```

```
class LinAttentionBridgeLayer(BaseAttentionBridgeLayer):
        return True
```

```
class SimpleAttentionBridgeLayer(BaseAttentionBridgeLayer):
        )
```

```
# TODO: for now I've used the basic implementation of TransformerEncoderLayer;
# we could consider an attention-bridge-specific implementation that would allow
# us to control the norm and return alphas if necessary.
class TransformerAttentionBridgeLayer(BaseAttentionBridgeLayer, TransformerEncoderLayer):
        )
```

```
class FeedForwardAttentionBridgeLayer(BaseAttentionBridgeLayer):
        )
```

straightforward configuration

```
ab_layers: ['feedforward', 'lin', 'transformer']
hidden_ab_size: 512
ab_fixed_length: 50
```

B) Bridges and structures for sharing

We saw stacks, we also have:

- bridges, shared across all tasks
- **adapters** for finer-grained sharing

```
class TransformerAdapterMixin:
    return result
```

```
class AdaptedTransformerEncoder(TransformerAdapterMixin, TransformerEncoder):
    return result
```

```
class AdaptedTransformerDecoder(TransformerAdapterMixin, TransformerDecoder):
    return result
```

with a highly flexible config

```
adapters:
  encoder:
    enc_group:
      layer_stack_index: 0
      layers: [0, 1]
      hidden_size: 8
      ids:
        - foo
        - bar
    enc_highresource:
      layer_stack_index: 0
      layers: [0, 1]
      hidden_size: 8
      ids:
        - en
        - de
    enc_lowresource:
      layer_stack_index: 0
      layers: [0]
      hidden_size: 8
      ids:
        - uu
  decoder:
    dec_group:
```

config config config

B) Bridges and structures for sharing

Define language groups and shared components through a language distance matrix

lang	en	de	fr	zh
en	0	0.1	0.2	1
de	0.1	0	0.2	1
fr	0.2	0.2	0	1
zh	1	1	[illegible]	0

config config

Group 1: {en, de, fr}

Group 2: {zh}

```
group_idx = AgglomerativeClustering(
    n_clusters=n_groups,
    metric='precomputed',
    linkage='average',
    distance_threshold=cutoff_threshold,
).fit_predict(distance_matrix['data']).tolist()
groups = {lang: f'group{idx}' for lang, idx in zip(distance_matrix['header'], group_idx)}
```

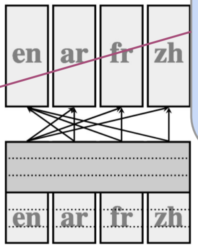
B) Bridges and structures for sharing

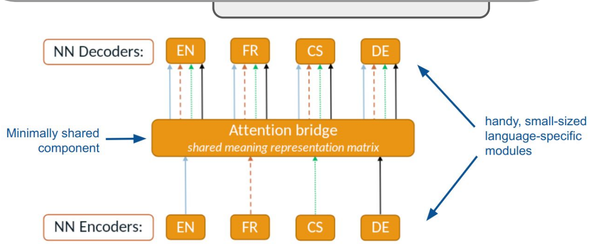
What structure we want to explore for the shared paramet

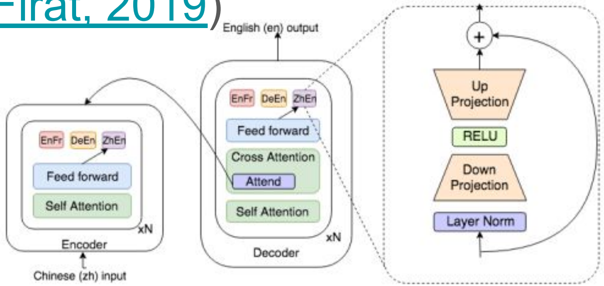
More coming soon!

This comparison is current ongoing work... we submitting to WMT!!!

- fully shared layers
  - Transformer-based
  - FFWD
  - Attention bridge (remember the lore?)
  - RNN, CNN?
- Adapters ([Bapna & Firat, 2019](#))







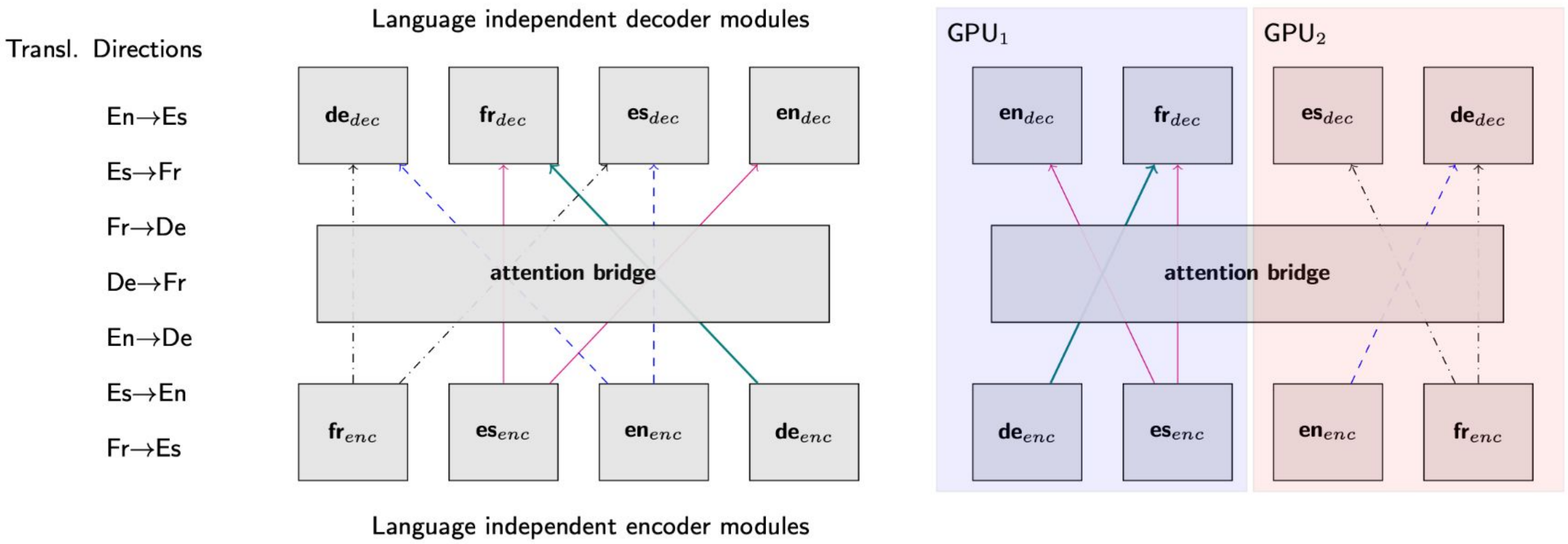
C) Communication chaos

Scaling-up a mNMT to a (very) large number of languages must:

- Deal with the task2gpu allocation problem
- Allow for custom model parallelism across nodes and GPUs

C) Communication chaos

- The task2gpu allocation problem



C) Communication chaos

- The task2gpu allocation problem in MAMMOTH*

1/ define costs for factors to weigh in

```
INTER_NODE_COST = 5
INTRA_NODE_COST = 1
HOMOGENEITY_WEIGHT = 0.1
READY_TO_START_WEIGHT = 0.5
UNASSIGNED_WEIGHT = 100
UNASSIGNED_PREFER_MASTER = 10
SPLIT_LPS_WEIGHT = 50
NOT_READY_TO_START = 500
VERY_BAD = 99999999
```

2/ randomly assign language pairs to gpu slots

** currently a beta feature*

C) Communication chaos

3/ Move all pairs to their best possible option

4/ rinse and repeat until stabilization

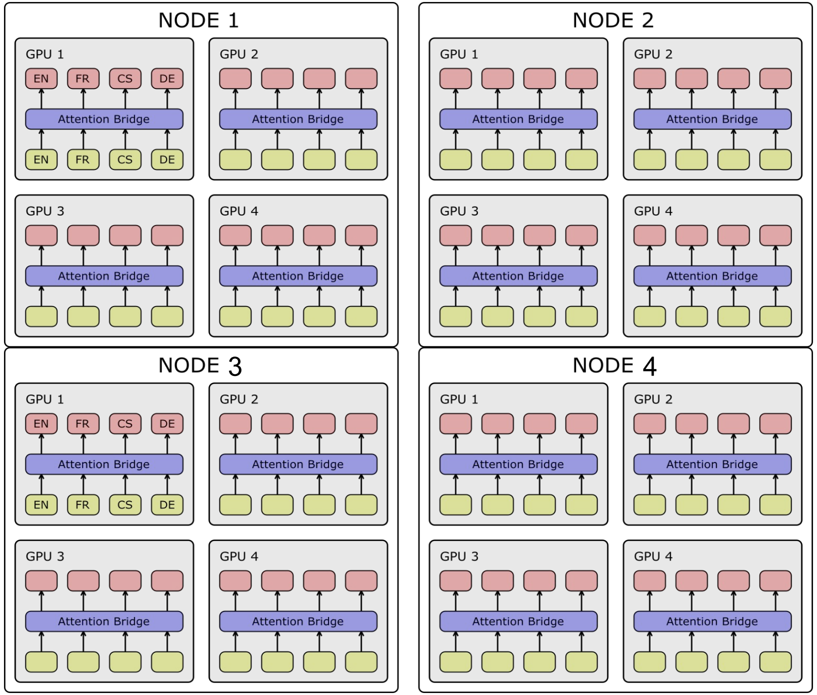
config config

```python
def swap_all_slots_once(self, assignment, current_cost):
    for i, slot_a in enumerate(self.gpu_slots):
        current_cost, assignment = self.best_swap_for(slot_a, assignment, current_cost)
        print('o', end='', flush=True)
    return current_cost, assignment

def optimize(self, assignment, current_cost, iterations=10, patience=1):
    prev_cost = None
    stalled = 0
    print(f'initial cost: {current_cost}', flush=True)
    for i in range(iterations):
        prev_cost = current_cost
        current_cost, assignment = self.swap_all_slots_once(assignment, current_cost)
        print(f'iteration {i} cost: {current_cost}', flush=True)
        if prev_cost == current_cost:
            stalled += 1
        else:
            stalled = 0
        if stalled > patience:
            print('No improvement, finishing early', flush=True)
            break
    return current_cost, assignment
```

C) Communication chaos

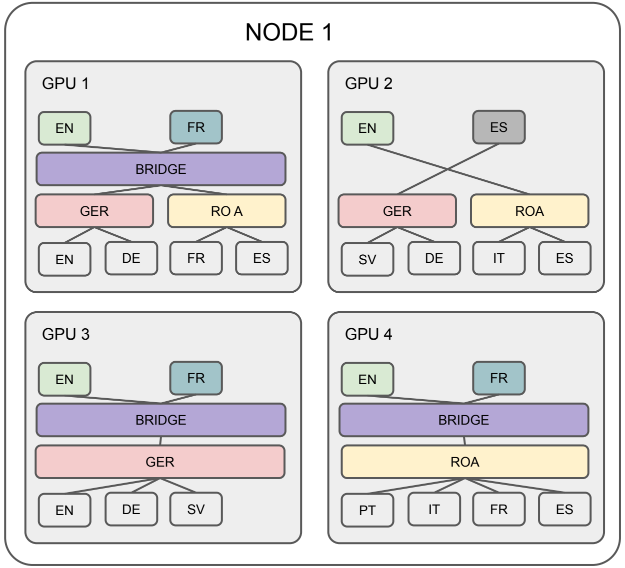
Custom model parallelism across nodes and GPUs



- In short, modules allocated in more than 1 GPU have to be synced at all times.

C) Communication chaos

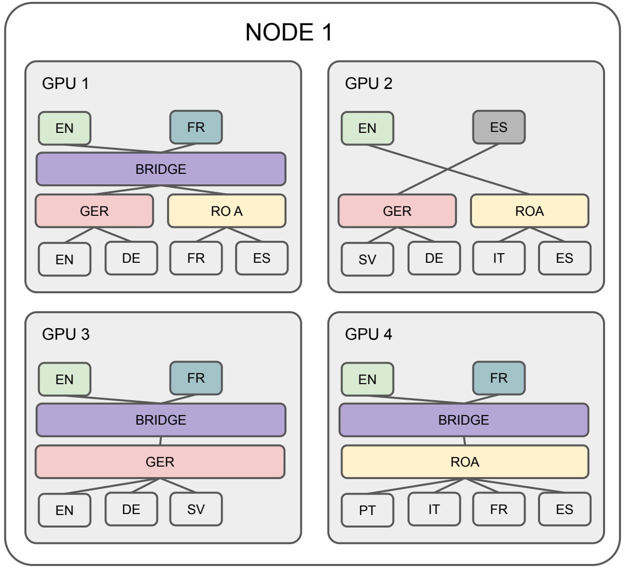
Custom model parallelism increases param. sharing versatility



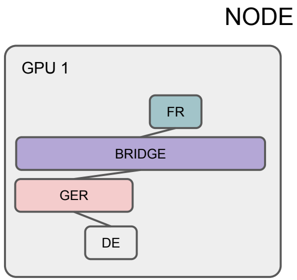
- Modules are synchronized in the GPUs where they are present:
  - AB layer synced in GPUs 1,3&4
  - Language-specific components synced as needed (e.g., EN-decoder in all GPUs)
  - Language group-specific components also synced as needed (e.g., GER in GPUs 1,2&3)

C) Communication chaos

Custom model parallelism increases param. sharing versatility **& inference efficiency**



- All modules are saved independently
- Light inference. E.g., DE->FR only loads



C) Communication chaos

At training time, *coder communication is based on layer stacks (and adapters)

```
self._gradient_accumulation_over_lang_pairs(
    batches_with_meta,
    total_stats,
    report_stats,
)

# Note that all group ids are tuples, some with length 1
for (layer_stack_index, encoder_id), (_, group) in self.my_encoder_groups.items():
    params = [
        (name, p) for (name, p)
        in self.model.encoder.get_submodule(layer_stack_index, encoder_id).named_parameters()
        if 'embeddings' not in name and 'adapter' not in name
    ]
    onmt.utils.distributed.only_ready_reduce_and_rescale_grads(params, group=group)

for (layer_stack_index, decoder_id), (_, group) in self.my_decoder_groups.items():
    params = [
        (name, p) for (name, p)
        in self.model.decoder.get_submodule(layer_stack_index, decoder_id).named_parameters()
        if 'embeddings' not in name and 'adapter' not in name
    ]
    onmt.utils.distributed.only_ready_reduce_and_rescale_grads(params, group=group)
```

We have tested comms in CUDA and ROCM!!*

* *Thanks to CSC infrastructure :)*

ICT Solutions for Brilliant Minds

CSC

C) Communication chaos

We only broadcast the gradient for modules currently in use

We have tested comms in CUDA and ROCM!!*

```python
ready_list = []
for name, p in require_grad:
    if hasattr(p, 'has_grad') and p.has_grad:
        ready_list.append(1.0)
    else:
        ready_list.append(0.0)
        if p.grad is None:
            p.grad = torch.zeros_like(p)

# Communicate the ready bits, and reduce them using summation.
# This gives the number of non-dummy gradients participating, for normalization
ready_t = torch.tensor(ready_list).to(device)
if group is None:
    torch.distributed.all_reduce(ready_t)
else:
    torch.distributed.all_reduce(ready_t, group=group)
rescale_denoms = ready_t  # after reduction
```

** Thanks to CSC infrastructure :)*

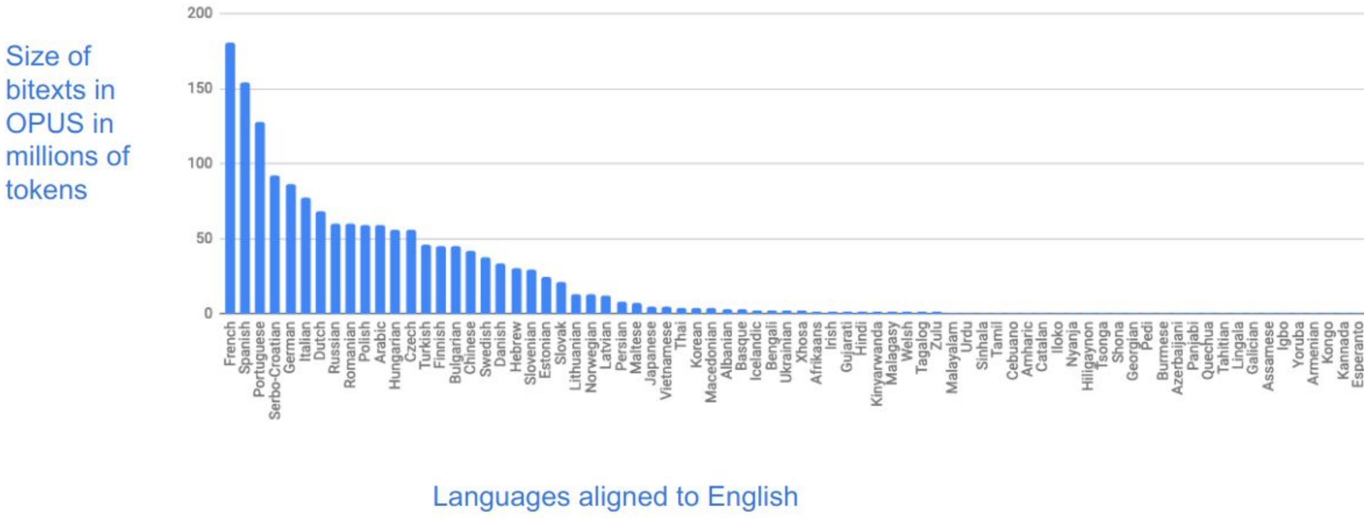
CSC

ICT Solutions for Brilliant Minds

D) Dearth of data

Remember?

(1) Limits of training data



D) Dearth of data

- Complex sharing structures in multi-node & -GPU settings make random task scheduling tricky
- Schedulers for
  - Delaying the start of specific tasks (start with overrepresented langpairs)
  - Early stopping of specific tasks (to better fine-tune underrepresented langpairs)
- Sampling schemes
  - Weight-based (sample more from high-res langpairs) / Round-robin based
  - More to come

Why use it?

MAMMOTH has been tailored for (massively) multilingual NMT:

1. Scales well to many tasks

2. Allows different param. sharing structures

3. It is modular ⇒ ☑️ big at training time
   ☑️ inference is lightweight

Sneak peek

Our MTM23 Project: Integration to HF 🤗

Convert MAMMOTH models to HF transformer models (idk, BART, Marian or T5 architecture):

- Convert a Mammoth task configuration snippet into a HF MarianConfig
- Construct a HF checkpoint from a Mammoth checkpoint and a task configuration snippet
- Plug-and-play utilities for converting modular Mammoth models within HF Transformers

Any question?

erc

FOTRAN
Found in Translation

HELSINGIN YLIOPISTO
HELSINGFORS UNIVERSITET
UNIVERSITY OF HELSINKI

NVIDIA®

SILO.AI

CSC
ICT Solutions for Brilliant Minds

MAMMOTH goes huggingface

Convert MAMMOTH models to HF transformer models (idk, BART, Marian or T5 architecture):

- Convert a Mammoth task configuration snippet into a HF MarianConfig
- Construct a HF checkpoint from a Mammoth checkpoint and a task configuration snippet
- Plug-and-play utilities for converting modular Mammoth models within HF Transformers

ICT Solutions for Brilliant Minds
CSC

erc

FOTRAN
Found in Translation

MAMMOTH goes huggingface

Convert MAMMOTH models to HF transformer models (idk, BART, Marian or T5 architecture):

- Convert a Mammoth task configuration snippet into a HF MarianConfig
- Construct a HF checkpoint from a Mammoth checkpoint and a task configuration snippet
- Plug-and-play utilities for converting modular Mammoth models within HF Transformers

CSC ICT Solutions for Brilliant Minds

erc

FOTRAN Found in Translation