Microsoft Research

Microsoft

Bimodal Program Analysis … in the Wild

Marc Brockschmidt (and Miltos Allamanis and many more)

Target: Variable Names

- Variable names are means of communication between SDE
  $\Longrightarrow$ natural language!

- Variable names are summaries of program semantics
  $\Longrightarrow$ static analysis!

Related Work

- Allamanis et al.:
  n-gram & LBL models for method / class / variable naming

- JSNice (Raychev et al.):
  Conditional random fields for names / types

Modern Times: 2017

- Input: Source code (fragments) around variable
- Output: Name

- Idea: Cast as translating code to name, use neural NLP toolbox:
  - Deep sequence to sequence architecture
  - Attention
  - Copying for out-of-vocabulary

Modern Times are Complex

Input comes in many forms:

- Tokens around variable uses:

```
var ??? = user.GetData();
```

- Types implemented by variable:

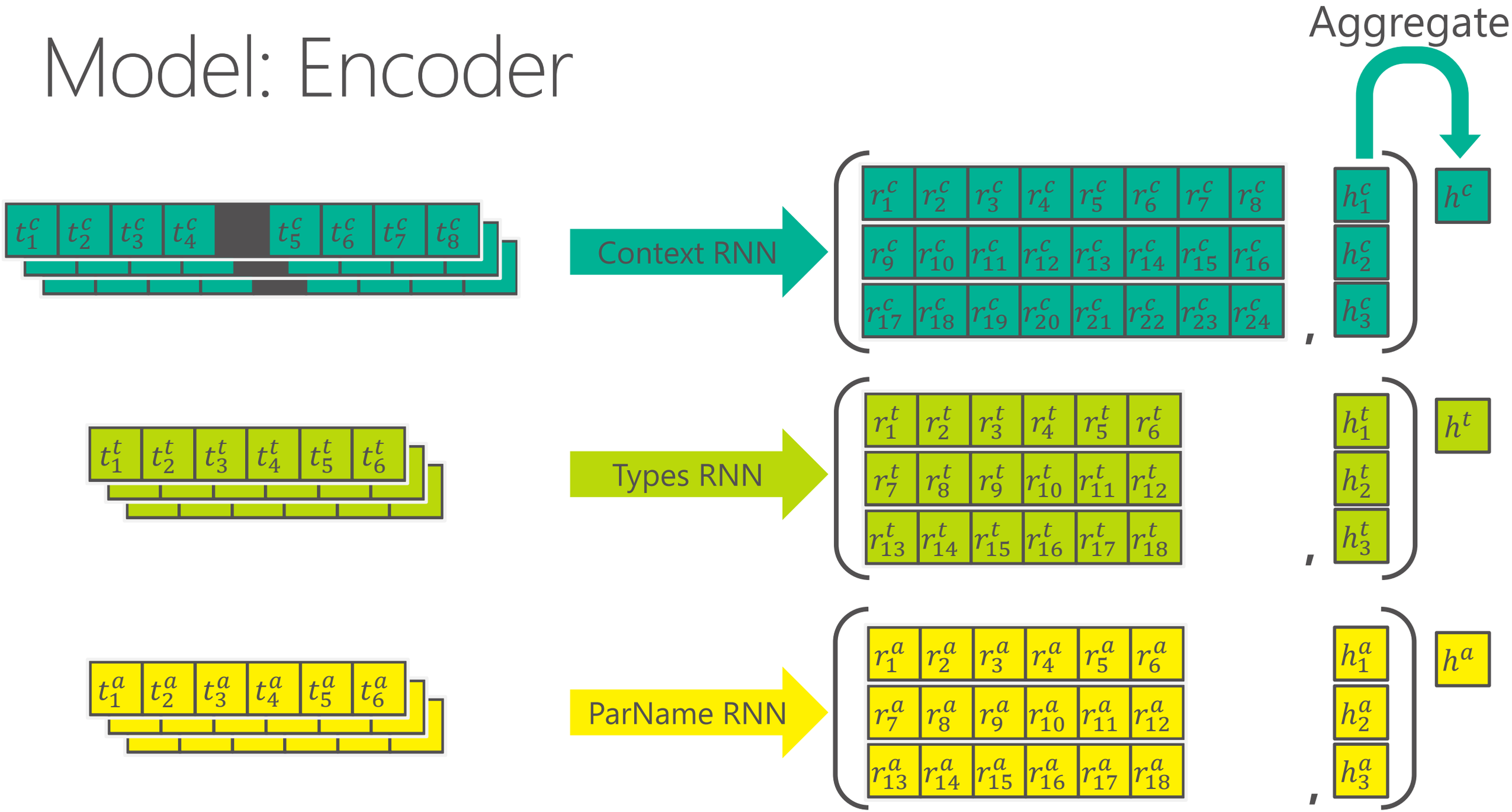
```
InputStream ??? = FooBar();
```

- Formal parameters it's matched to:

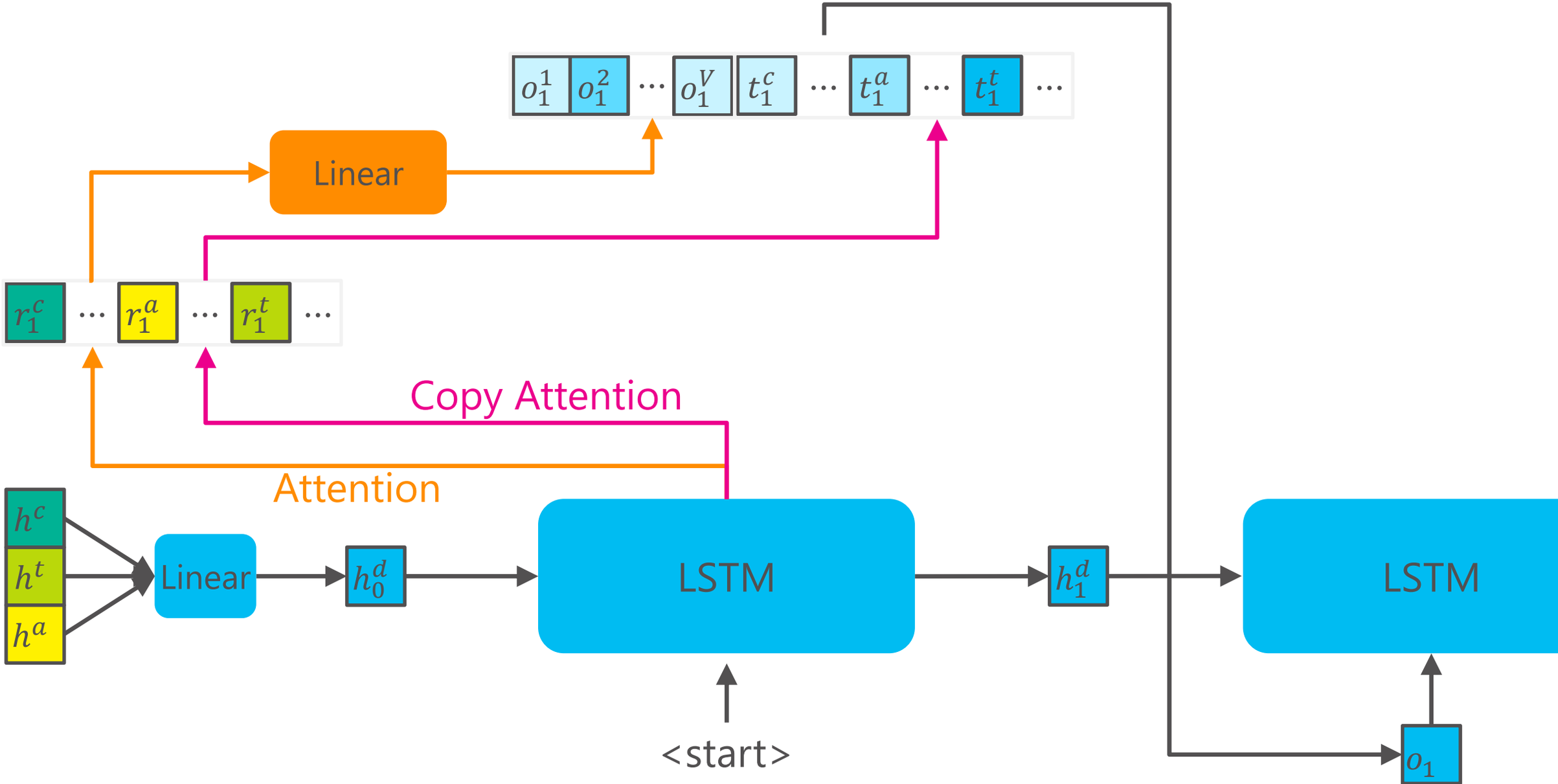
```
foo(???);
…
void foo(int userNum) { … }
```

⟹ Model needs to combine different information sources

Model: Encoder



Model: Decoder



Evaluation

Model	EF Acc.	ravenDB Acc.	EF Substr. Acc.	ravenDB Substr. Acc.
Plain LSTM Decoder	44%	38%	59%	50%
+ Attention	43%	42%	58%	55%
+ Copying	47%	44%	64%	55%

[on EntityFramework & RavenDB (medium/large C# OSS projects)]

Results – Group I: Be specific

```
}
catch (ErrorResponseException e)
{
    if (e.StatusCode == HttpStatusCode.NotFound)
    {
        var text = e.ResponseString;
        if (text.Contains("maxQueryString"))
            throw new ErrorResponseException(e, text);

                                                tion(e, "There is no index named: " + index);

}

if (HandleException(responseException))
    return null;
```

Model	Suggestions
Suggestions	message (53.87%),

Results – Group I: Be specific

```
var documentStores = new Dictionary<string, IDocumentStore>
                            {
                                {"one", new DocumentStore{Url = "http://localhost:8078"}},
                                {"two", new DocumentStore{Url = "http://localhost:8077"}},
                                {"tri", new DocumentStore{Url = "http://localhost:8076"}},
                            };

foreach (var documentStore in documentStores)
{
    documentStore.Value.Conventions.FailoverBehavior = FailoverBehavior.FailImmediately;
}


var shardStrategy = new ShardStrategy(documentStores)
```

Model	Suggestions
Suggestions	shards (75.71%),

```
stId);

trategy);
```

Results – Group II: Be descriptive

```
if (fieldsToFetch.FetchAllStoredFields)
{
    var fields = new HashSet<string>(document.GetFields().Select(x => x.Name));
    fields.Remove(Constants.ReduceKeyFieldName);
    var documentFromFields = new RavenJObject();
    AddFieldsToDocument(document, fields, documentFromFields);
```

Model	Suggestions
Suggestions	fieldNames (64.59%),

```
lue;
}
```

Results – Group II: Be descriptive

```
public BoundParameter Parameter(ParameterSymbol p)
{
    return new BoundParameter(Syntax, p, p.Type) { WasCompilerGenerated = true };
}

publi                                                    eceiver, FieldSymbol f)
{
    r                                                    er, f, ConstantValue.NotAvailable, LookupResultKind.Viable, f.
}
```

Model	Suggestions
Suggestions	parameter (67.45%),

Models? What are they good for?

Deep Learning models are awesome!

1. … but need so much data!
2. … and also need to learn project-specific customs!
3. … and need a GPU for training (and during inference?)
4. … inference too slow for live feedback anyway
5. … so when do we show feedback to users?

Training in the Wild

Most projects (start) small

$\Longrightarrow$ Pre-train on corpus of projects, adapt to local project

Technically:

- Leave parts of decoder vocabulary unused in pre-training
- Fill up with project-specific data
- Retrain all parameters on project data

Consequences:

- Pre-training is expensive (lots of data, many epochs required)
- Adaptation is cheap (little data, few epochs required)

Training and Inference in the Wild

Deep RNN models require training, are big, slow on CPU

$\Longrightarrow$ Training and inference as a service

Technically:

- RESTful API, selecting model & providing variable contexts
- Server trains, stores and runs models (no deployment troubles!)
- Clients extract relevant data and do UI

Consequences:

- Deploying new models becomes easy
- Clients are thin, can be integrated into everything

UX in the Wild

Idea: Warn if certain that name should differ from used

Analysis requires some context, too slow/expensive for live warnings

$\Longrightarrow$ Do in CI, PR or on request

Questions:

- How to use user feedback (action/no action)?
- How to avoid repeating warnings?
- How to decide how aggressively to show warnings?

Conclusions

The good:

- Neural NLP toolbox can work for source code as well
- Effective models possible, can catch interesting issues

The bad:

- Deep Learning models hard to deploy, relatively expensive to train
- Infrastructure needs to be re-invented all the time

The ugly:

- UX aspects not studied, research focused on headline numbers
- Trade-off between model complexity & performance unclear