ModelMate: A recommender for textual modeling based on pre-trained language models

Jesús Sánchez Cuadrado

Universidad de Murcia

jesusc@um.es

@sanchezcuadrado

http://github.com/jesusc

http://sanchezcuadrado.es

KOJ0

ProxyHands

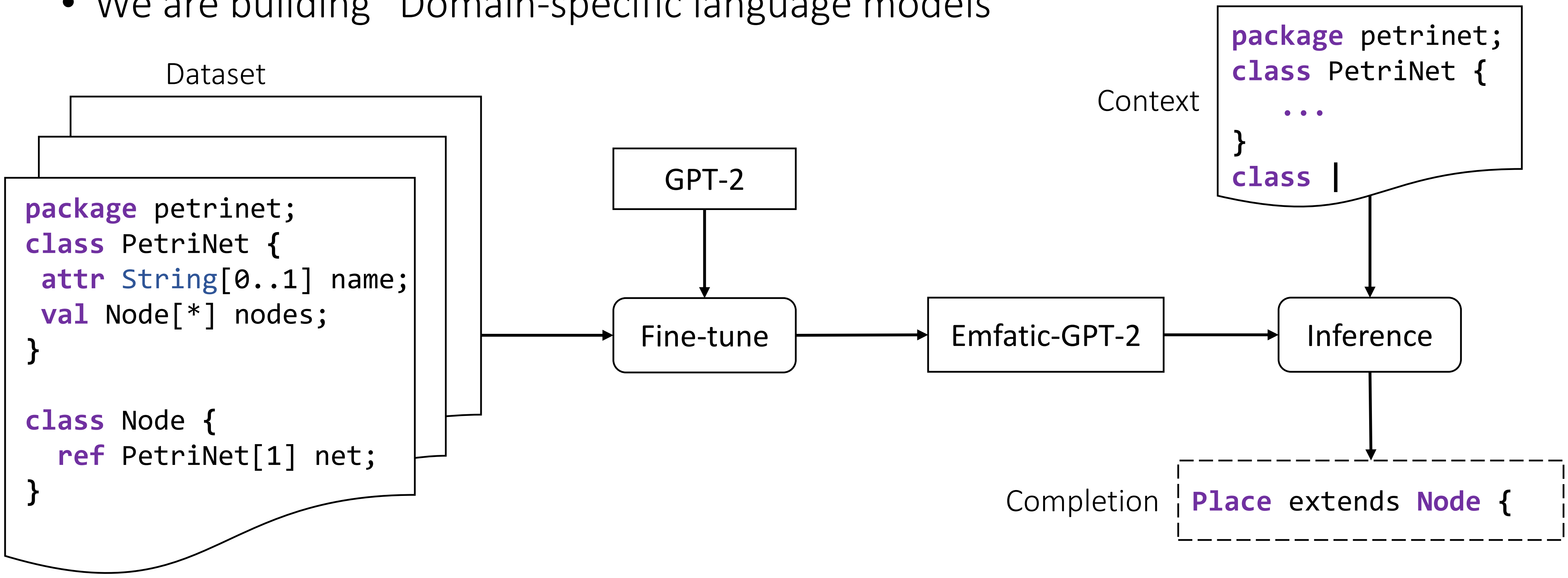
Motivation

- **It would be great to have a competitive recommender for DSLs**
  - IDEs for programmings languages have very powerful assistants
  - Current trend boosted by LLMs
  - Well known example: Copilot

- In modelling, there are many limitations
  - Built on top of small datasets
  - Built for just one language (typically Ecore)
  - Bad performance or not well-tested
  - Not released as an usable package
  - Do not work for textual languages
    - Why? Because they assume the whole model is in memory

Proposal

- Given a DSL, build a recommender by adapting a pre-trained language model to handle programs written in this DSL.
  - We are building "Domain-specific language models"



Recommendation **tasks**

- Identify relevant tasks supported by the recommender
- In this work:
  - Fragment completion
  - Line completion
  - Identifier suggestion

Tasks – **Fragment completion**

Given a context…

```
@namespace(uri="http://pn", prefix="pn")
package petrinet;

class PetriNet {
```

Tasks – Fragment completion

Given a context…

Complete the current fragment up to a relevant place

```
@namespace(uri="http://pn", prefix="pn")
package petrinet;

class PetriNet {
  attr String[0..1] name;
  val Node[*] nodes;
  val Transition[*] transitions;
}
```

The most general type of completion, can be triggered at any time

Tasks – **Line completion**

Given a context…

```
@namespace(uri="http://pn", prefix="pn")
package petrinet;

class PetriNet {
  attr String[0..1] name;
  
```

Tasks – Line completion

Given a context…

Generate a line

```
@namespace(uri="http://pn", prefix="pn")
package petrinet;

class PetriNet {
  attr String[0..1] name;
  val Node[*] nodes;
```

Typically corresponds to a complete language construct.

Tasks – Identifier suggestion

Given a context, up to a certain syntactic point…

```
@namespace(uri="http://pn", prefix="pn")
package petrinet;

class Place {
  attr String[1] name;
}

class PetriNet {
  val
```

Tasks – Identifier suggestion

Given a context, up to a certain syntactic point…

Generate suggestions

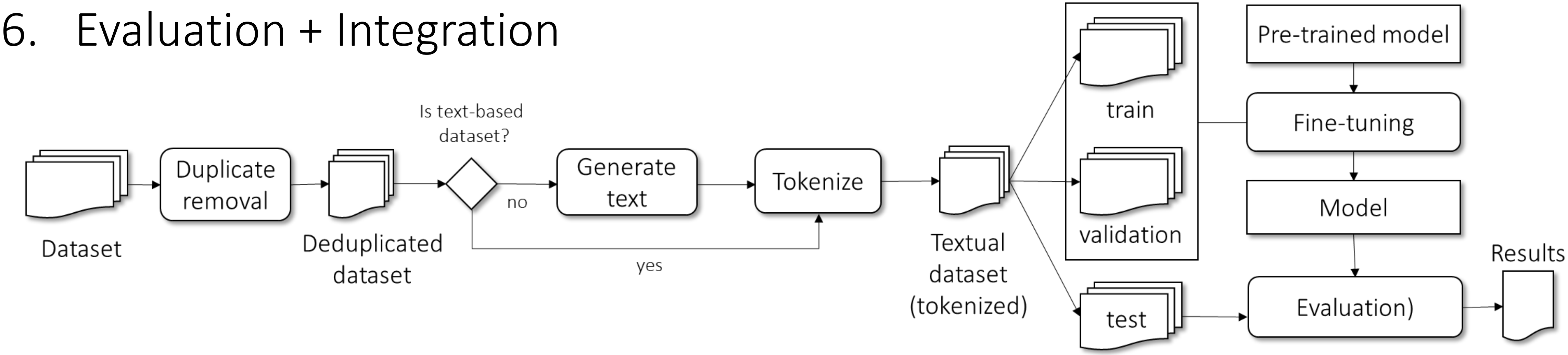
```
@namespace(uri="http://pn", prefix="pn")
package petrinet;

class Place {
  attr String[1] name;
}

class PetriNet {
  val
```

suggestions

Place
Transition
Arc

Approach – **Overview**

1. Dataset selection
2. Duplication removal
3. Text generation and tokenization
4. Model selection
5. Fine-tuning (training)
6. Evaluation + Integration



Approach – **Datasets**

- Textual dataset available
  - It is possible to obtain many textual files
  - E.g., Xtext, PlantUML
- Serialized dataset available
  - We have the XMIs
  - Typically when the DSL has several notations
  - E.g., Ecore/Emfatic, Ecore/OclInEcore
- No dataset available, but compatible
  - Typically for new DSLs (cold-start problem)
  - Transform a semantically compatible dataset into textual files

Approach – **Tokenization**

- **Tokenize** the input using the standard tokenizer of the language
  - Or at the same that text is generated (we provide an API for this)
- Tokenization facilitates the evaluation and post-processing of the inference
- Makes the model a bit more robust because the text is normalized

```
<s> package petrinet ; <EOL> class PetriNet { <EOL>
attr String [ 0 ] name ; ... } </s>
```

Approach – **Text generation and tokenization**

- If the dataset is not text-based, build a model-to-text (serializer)
  - Allows for the adaptation of semantically-compatible datasets
  - Typically straightforward (4 – 6 hours) using a simple fluent API

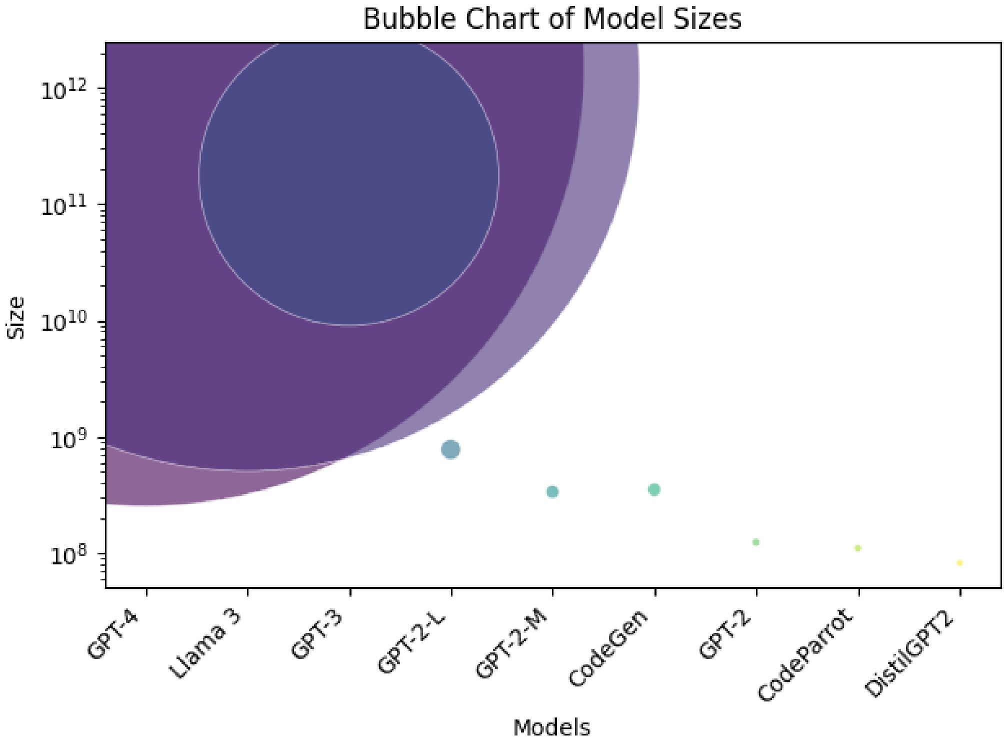
```
Class umlClass = ...;

output.token("entity").token(toName(umlClass.getName())).w();

if (element.getGenerals().size() > 0) {
    output.token("extends").w();
    output.token(toName(umlClass.getGenerals().get(0).getName())).w();
}

output.token("{").newLine();
// Transform attributes
output.token("}");
```

Approach – **Models**

- Prefer small PLMs

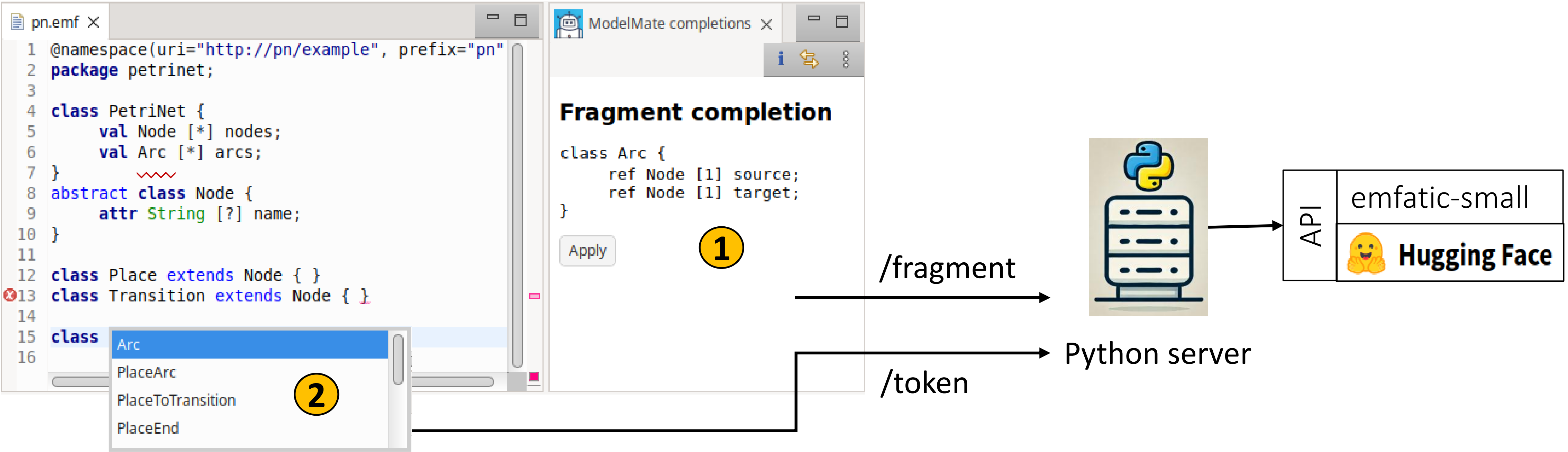
PLM	Parameters	Pre-training data
GPT2 [23]	124M	WebText [23]
GPT2-M [23]	355M	WebText [23]
GPT2-L [23]	774M	WebText [23]
DiltilGPT2 [25]	82M	OpenWebText [22]
CodeParrot-small-multi	110M	CodeParrot dataset
Codegen-nl [20]	350M	The Pile [10]
Codegen-multi [20]	350M	BigQuery [20]
Codegen-mono [20]	350M	BigPython [20]

Table 1: PLMs considered in ModelMate.



Sneak peak: CodeParrot (7 times smaller than GPT-2-L) after fine-tuning is as good as GPT-3.5 (akin to GPT-4)

ModelMate – Implementation and availability



Checkout https://github.com/models-lab/model-mate

Demo time!

Evaluation

1. Accuracy of ModelMate in the recommendation tasks for three different DSLs
   - Emfatic, Xtext, Entities
   - Showcase the three scenarios of dataset availability
2. Comparison against state-of-the-art model recommenders
   - Feature recommendation
   - EcoreBert, MemoRec, KNN/Glove
3. Comparison against LLMs
   - Use in-context learning capabilities by showing autocompletion examples
   - Use GPT3.5-turbo-instruct (similar to GPT-4 but cheaper)
4. Inference time
   - Small vs. large models

Evaluation – **Assessment**

- Overall performance – Good
  - Emfatic – 69%
  - Xtext – 70%
  - Entity – 77%

- ModelMate outperforms existing recommenders
  - Generalizes better
  - (the experiment with the MAR/GenMyModel dataset is a hard problem)
    - SR@5 = 0.18 vs 0.9

- ModelMate vs LLMs
  - On par with GPT-3.5 which has several orders of magnitude more parameters
  - ModelMate can be executed in a modest GPU or even on CPU

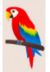
Evaluation – **Assessment**

- Impact of the syntax in the performance
  - The context for the name is better than for the type

```
class PetriNet {
  val |
```

```
class PetriNet {
  val Node[*] |
```

  - Take this into account when you design your DSL

- PLM selection
  - The cost associated with building a DSL should be modest
    - This also applies to training smart AI facilities and running them (locally)
  - Choose the model with the best accuracy/cost
    - codeparrot 

Evaluation – **Assessment**

- Practical usage – what if you have a new DSL
  - Find out how to obtain a dataset
    - The key is to have a compatible dataset
    - We introduces the notion of semantically compatible dataset
    - Deriving a dataset from an existing one via model-to-text transformation: around 4-6 hours
  - Fine-tune the models
    - We provide a simple framework based on configuration files
  - Integrate in Eclipse
    - We provide a base plug-in
- Practical use – how it works for the user
  - We found it smooth (good inference time) and with relevant suggestions
  - Provides suggestions even when the text has errors (our goal!)

Conclusions

- ModelMate: building smart editing facilities for textual DSLs
  - Complete workflow easily adaptable to other DSLs
  - Support for three different tasks
  - Evaluated with three DSLs.
    - ModelMate consistently achieves good performance
    - On-par with GPT-3.5
  - Better generalization capabilities than other model recommenders
  - Eclipse plug-in available, Python implementation

- Future work
  - Publish an update site!
  - Use LLMs to build other tasks without compromising performance
  - Enrich datasets to enhance their quality
  - Take into account the text after the cursor
  - Better integration of the generated code (pretty printing, syntax errors, etc.)

KOJ0

ProxyHands

Thanks!

Checkout and give us feedback

https://github.com/models-lab/model-mate

MODELS & LANGUAGES LAB

http://models-lab.github.io

jesusc@um.es

@sanchezcuadrado

http://github.com/jesusc

http://sanchezcuadrado.es

Evaluation – Next token prediction (general accuracy)

Model	Accuracy	Identifier Suggestion						Line		Fragment
		class name	super name	attr. type	ref. type	val. type	feature name	EM	Edit Similarity	BLEU
codegen-mono	69.10	45.03	78.28	77.24	52.91	39.22	59.35	14.27	50.51	31.21
codegen-multi	69.07	45.19	78.33	77.73	53.45	39.03	59.36	14.37	50.81	31.32
codeparrot	68.64	44.25	77.65	77.72	50.84	37.79	58.35	13.90	48.63	30.67
gpt2-large	68.52	42.29	75.41	76.60	50.58	36.87	57.51	13.24	48.53	29.93
codegen-nl	68.02	41.55	76.31	75.75	50.21	35.87	57.23	12.06	46.04	29.11
gpt2-medium	67.65	39.97	74.24	75.91	48.34	33.66	54.83	11.63	44.36	28.39
gpt2	65.83	35.19	70.78	74.49	42.15	28.77	51.07	9.16	41.76	26.15
distil-gpt2	64.02	29.08	66.30	73.15	36.65	23.58	47.02	6.48	37.19	24.09

- `codegen-mono/multi` is generally the best model
- `codeparrot` is on par
- Good general accuracy in token prediction (almost 70%)

Evaluation – **Identifier suggestion**

		Identifier Suggestion						Line		Fragment
Model	Accuracy	class name	super name	attr. type	ref. type	val. type	feature name	EM	Edit Similarity	BLEU
codegen-mono	69.10	45.03	78.28	77.24	52.91	39.22	59.35	14.27	50.51	31.21
codegen-multi	69.07	45.19	78.33	77.73	53.45	39.03	59.36	14.37	50.81	31.32
codeparrot	68.64	44.25	77.65	77.72	50.84	37.79	58.35	13.90	48.63	30.67
gpt2-large	68.52	42.29	75.41	76.60	50.58	36.87	57.51	13.24	48.53	29.93
codegen-nl	68.02	41.55	76.31	75.75	50.21	35.87	57.23	12.06	46.04	29.11
gpt2-medium	67.65	39.97	74.24	75.91	48.34	33.66	54.83	11.63	44.36	28.39
gpt2	65.83	35.19	70.78	74.49	42.15	28.77	51.07	9.16	41.76	26.15
distil-gpt2	64.02	29.08	66.30	73.15	36.65	23.58	47.02	6.48	37.19	24.09

- Some identifiers like super class names and attribute types are easier
    - Very good performance

Evaluation – **Identifier suggestion**

Model	Accuracy	Identifier Suggestion						Line		Fragment
		class name	super name	attr. type	ref. type	val. type	feature name	EM	Edit Similarity	BLEU
codegen-mono	69.10	45.03	78.28	77.24	52.91	39.22	59.35	14.27	50.51	31.21
codegen-multi	69.07	45.19	78.33	77.73	53.45	39.03	59.36	14.37	50.81	31.32
codeparrot	68.64	44.25	77.65	77.72	50.84	37.79	58.35	13.90	48.63	30.67
gpt2-large	68.52	42.29	75.41	76.60	50.58	36.87	57.51	13.24	48.53	29.93
codegen-nl	68.02	41.55	76.31	75.75	50.21	35.87	57.23	12.06	46.04	29.11
gpt2-medium	67.65	39.97	74.24	75.91	48.34	33.66	54.83	11.63	44.36	28.39
gpt2	65.83	35.19	70.78	74.49	42.15	28.77	51.07	9.16	41.76	26.15
distil-gpt2	64.02	29.08	66.30	73.15	36.65	23.58	47.02	6.48	37.19	24.09

- Some identifiers like super class names and attribute types are easier
  - Very good performance
- Class names, references and feature names are more difficult
  - Still, relatively good performance (around 50%)

Evaluation – **Identifier suggestion**

		Identifier Suggestion						Line		Fragment
Model	Accuracy	class name	super name	attr. type	ref. type	val. type	feature name	EM	Edit Similarity	BLEU
codegen-mono	69.10	45.03	78.28	77.24	52.91	39.22	59.35	14.27	50.51	31.21
codegen-multi	69.07	45.19	78.33	77.73	53.45	39.03	59.36	14.37	50.81	31.32
codeparrot	68.64	44.25	77.65	77.72	50.84	37.79	58.35	13.90	48.63	30.67
gpt2-large	68.52	42.29	75.41	76.60	50.58	36.87	57.51	13.24	48.53	29.93
codegen-nl	68.02	41.55	76.31	75.75	50.21	35.87	57.23	12.06	46.04	29.11
gpt2-medium	67.65	39.97	74.24	75.91	48.34	33.66	54.83	11.63	44.36	28.39
gpt2	65.83	35.19	70.78	74.49	42.15	28.77	51.07	9.16	41.76	26.15
distil-gpt2	64.02	29.08	66.30	73.15	36.65	23.58	47.02	6.48	37.19	24.09

- Some identifiers like super class names and attribute types are easier
  - Very good performance
- Class names, references and feature names are more difficult
  - Still, relatively good performance (around 50%)

Evaluation – **Line completion**

		Identifier Suggestion						Line		Fragment
Model	Accuracy	class name	super name	attr. type	ref. type	val. type	feature name	EM	Edit Similarity	BLEU
codegen-mono	69.10	45.03	78.28	77.24	52.91	39.22	59.35	14.27	50.51	31.21
codegen-multi	69.07	45.19	78.33	77.73	53.45	39.03	59.36	14.37	50.81	31.32
codeparrot	68.64	44.25	77.65	77.72	50.84	37.79	58.35	13.90	48.63	30.67
gpt2-large	68.52	42.29	75.41	76.60	50.58	36.87	57.51	13.24	48.53	29.93
codegen-nl	68.02	41.55	76.31	75.75	50.21	35.87	57.23	12.06	46.04	29.11
gpt2-medium	67.65	39.97	74.24	75.91	48.34	33.66	54.83	11.63	44.36	28.39
gpt2	65.83	35.19	70.78	74.49	42.15	28.77	51.07	9.16	41.76	26.15
distil-gpt2	64.02	29.08	66.30	73.15	36.65	23.58	47.02	6.48	37.19	24.09

- EM = Exact Match
  - 14% of the completions exactly the same
- Edit similarity = Levenshtein similarity
  - 50% is a good result

Evaluation – **Fragment completion**

		Identifier Suggestion						Line		Fragment
Model	Accuracy	class name	super name	attr. type	ref. type	val. type	feature name	EM	Edit Similarity	BLEU
codegen-mono	69.10	45.03	78.28	77.24	52.91	39.22	59.35	14.27	50.51	31.21
codegen-multi	69.07	45.19	78.33	77.73	53.45	39.03	59.36	14.37	50.81	31.32
codeparrot	68.64	44.25	77.65	77.72	50.84	37.79	58.35	13.90	48.63	30.67
gpt2-large	68.52	42.29	75.41	76.60	50.58	36.87	57.51	13.24	48.53	29.93
codegen-nl	68.02	41.55	76.31	75.75	50.21	35.87	57.23	12.06	46.04	29.11
gpt2-medium	67.65	39.97	74.24	75.91	48.34	33.66	54.83	11.63	44.36	28.39
gpt2	65.83	35.19	70.78	74.49	42.15	28.77	51.07	9.16	41.76	26.15
distil-gpt2	64.02	29.08	66.30	73.15	36.65	23.58	47.02	6.48	37.19	24.09

- BLEU computes n-gram overlapping between expected and generated
- Gives an idea of the quality of the produced text
    - 30 – 40% : Understandable to good translations
    - 40 – 40% : High-quality translations
    - 50 – 60% : Very high-quality and fluent translations

Evaluation – **Comparison against LLMs**

		Identifier Suggestion						Line		Fragment
Model	Accuracy	class name	super name	attr. type	ref. type	val. type	feature name	EM	Edit Similarity	BLEU
GPT-3.5	**80.58**	**42.00**	79.00	72.50	**58.50**	**51.5**	54.00	**22.60**	**65.80**	23.73
codegen-multi	69.24	41.23	78.93	80.49	52.21	37.71	**62.87**	14.20	48.84	**31.65**
codeparrot	68.71	41.83	**80.58**	79.78	50.23	38.60	62.72	13.10	46.54	30.87
gpt2-large	68.52	40.95	78.46	**81.35**	47.41	39.05	61.25	12.60	45.12	30.42

- ModelMate on par with GPT-3.5
  - GPT-3.5 only “wins” clearly in next token prediction and containment references
  - ModelMate “wins” clearly wins in attribute types, feature names, fragment completion.
- A small model like **codeparrot** is competitive

Evaluation – **Comparison against LLMs**

	ModelSet		MAR/GenMyModel	
	MRR@5	SR@5	MRR@5	SR@5
ModelMate	0.52	0.64	**0.18**	**0.25**
EcoreBert	0.34	0.47	0.09	0.13
MemoRec	**0.72**	**0.73**	0.10	0.12
KNN/Glove	0.70	0.75	0.06	0.08

- How well ModelMate performs
  - Task: The task is as follows: given an Ecore meta-model with one feature
  - removed, predict the name of the removed feature
- EcoreBert was trained on the MAR dataset
  - Train MemoRec and KNN/Glove with the same dataset
  - Important note: ModelSet is a subset of the MAR dataset
- Evaluate with ModelSet and an unseen dataset MAR/GenMyModel
  - ModelMate is able to generalize

Evaluation – **Inference time**

	Identifier	Line	Fragment
codeparrot	84.34 ± 40.0	130.2 ± 43.8	420.0 ± 714.0
codegen-multi	232.2 ± 115.6	341.9 ± 130.0	1490.7 ± 2311.6
gpt2-large	365.4 ± 217.8	425.6 ± 191.6	1877.8 ± 2801.5

Table 8: Inference time of ModelMate/Emfatic. Average time (ms) of 1,000 samples per task ± the standard deviation.

- Codeparrot is quite fast

Approach – **Training**

- Split the dataset into train-validation-test
- Train-validation with early stopping
  - Other hyperparameters like context=512

Approach – **Task evaluation procedure**

- Using the test set, derive concrete test sets for each task
  - Pair: <context>, <expected>
- Tasks:
  - Next token prediction
    - Basic measure of accuracy
  - Fragment completion
    - Identify the beginning of a block (e.g., {)
    - Context = text before the block, Expected = text up to the next closing (e.g., })
    - Metric: BLEU (measures n-gram overlapping)
  - Line completion
    - Find each <EOL>
    - Context = text before the <EOL>, Expected = text up to the next <EOL>
    - Metric: Exact Match and Edit distance
  - Identifier suggestion
    - Find each identifier of interest (e.g., class name)
    - Context = text before the identifier, Expected = the identifier
    - Metric: MRR@5