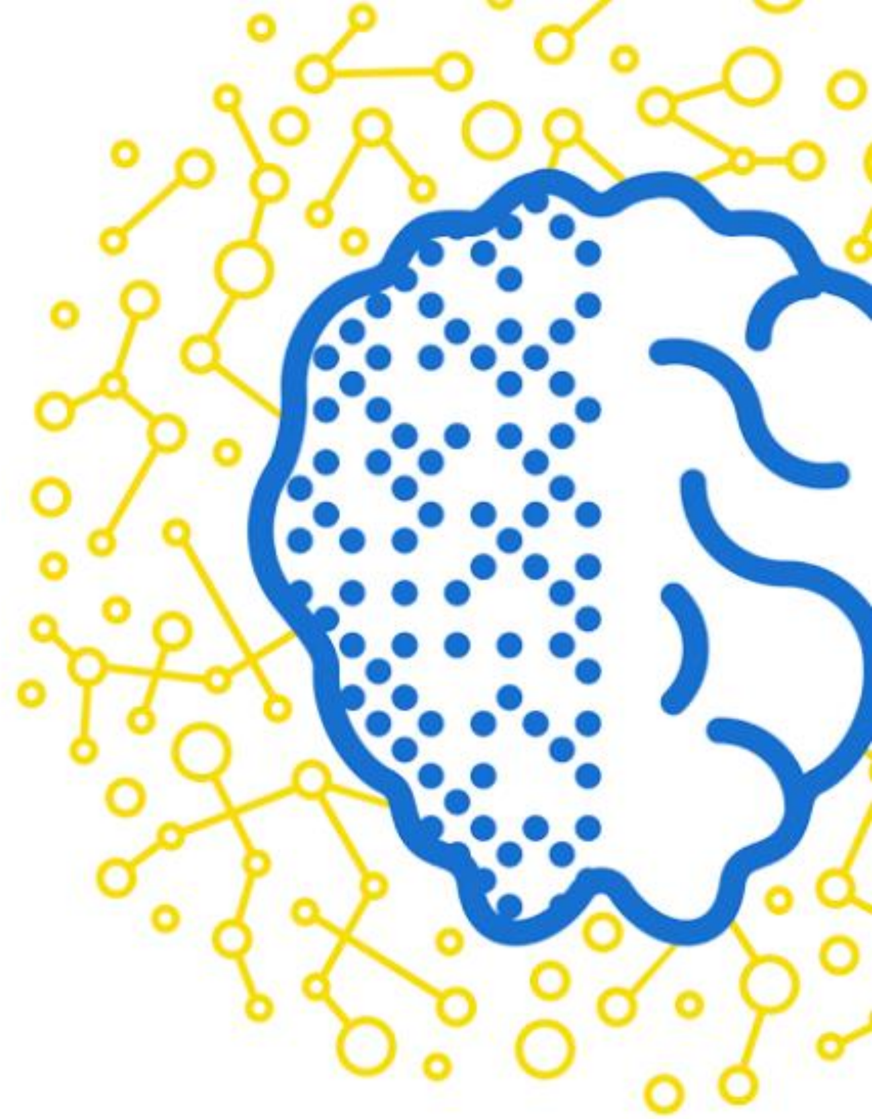
Introduction to DeepPavlov

Mikhail Burtsev, PhD

Moscow Institute of Physics and Technology (MIPT)

MIPT

MOSCOW INSTITUTE OF PHYSICS AND TECHNOLOGY



DeepPavlov

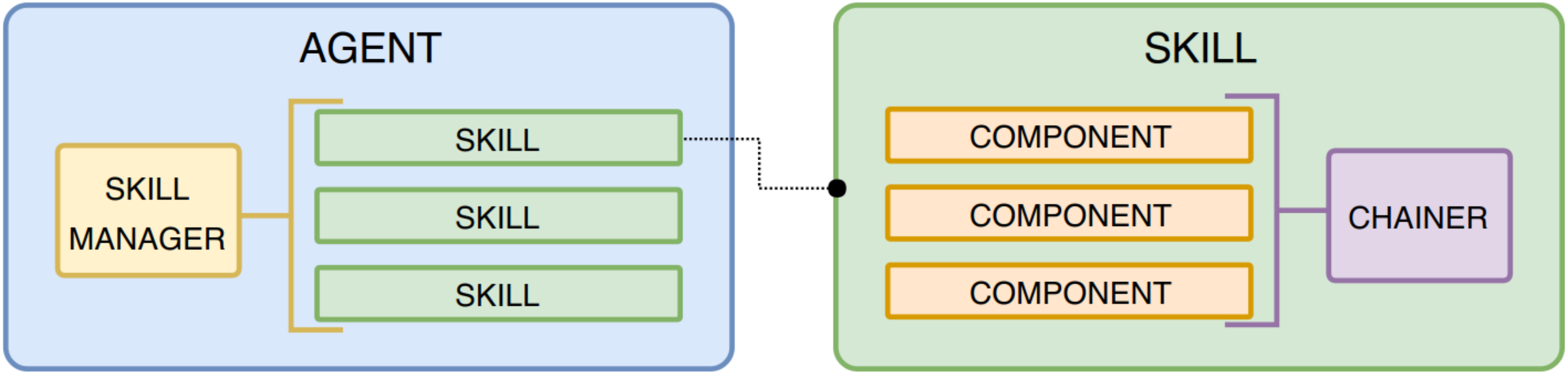
- DeepPavlov is for
  - **development** of production ready chat-bots and complex conversational systems,
  - NLP and dialog systems **research**.

- DeepPavlov’s goal is to enable AI-application developers and researchers with:
  - set of **pre-trained NLP models**, pre-defined dialog system components (ML/DL/Rule-based) and conversational **agents templates for a typical scenarios**;
  - a framework for **implementing** and testing their own **dialog models**;
  - tools for application **integration** with adjacent infrastructure (messengers, helpdesk software etc.);
  - **benchmarking** environment for conversational models and uniform access to relevant datasets.

Core concepts

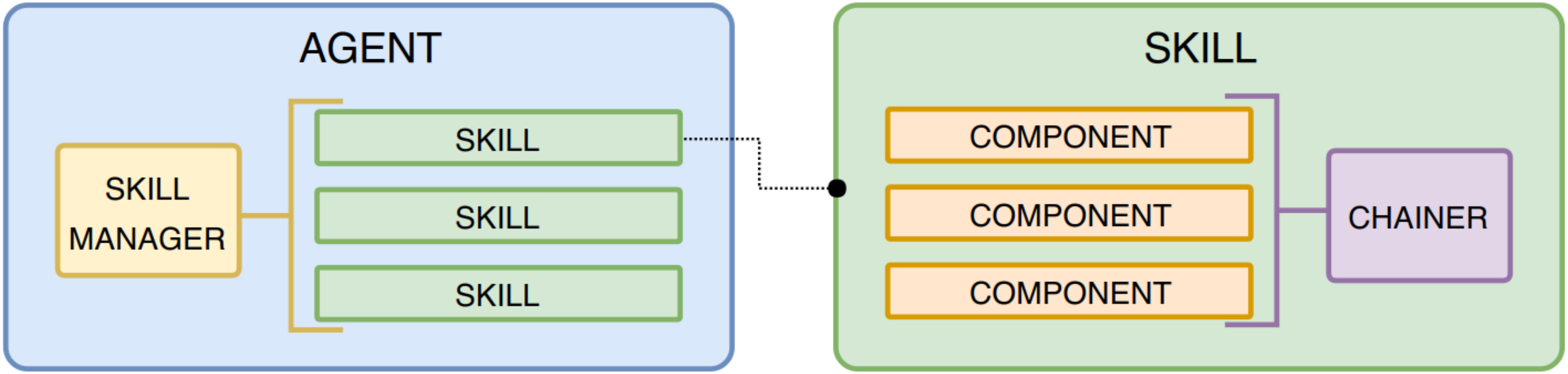
Dialogue system combines

complimentary skills to help user.



Core concepts

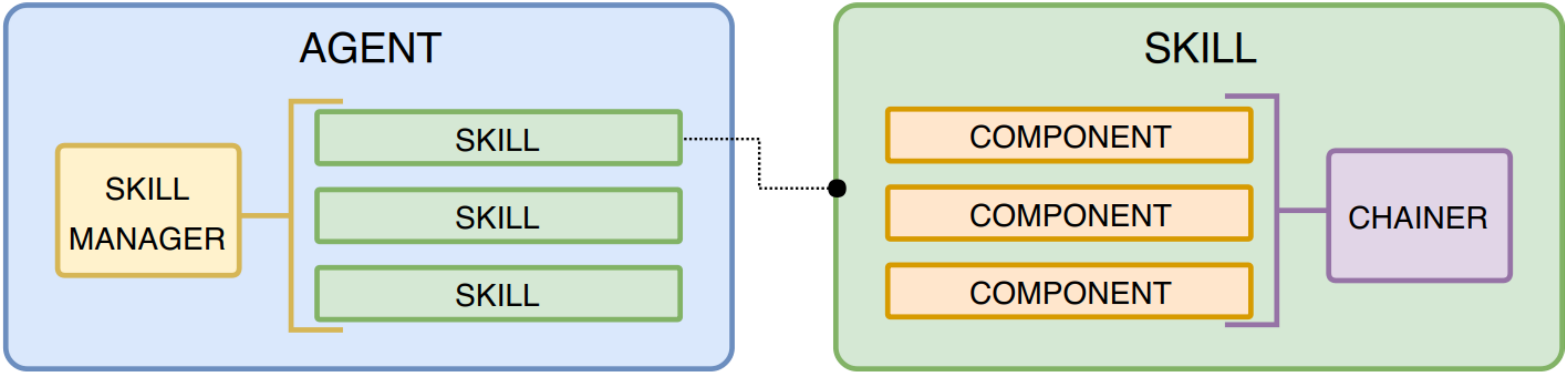
- `Agent` is a conversational agent communicating with users in natural language (text).
- `Skill Manager` performs selection of the `Skill` to generate response.
- `Skill` fulfills user’s goal in some domain. Typically, this is accomplished by presenting information or completing transaction (e.g. answer question by FAQ, booking tickets etc.). However, for some tasks a success of interaction is defined as continuous engagement (e.g. chit-chat).



Core concepts

- `Component` is a reusable functional component of `Skill`.

- `Chainer` builds an agent/component pipeline from heterogeneous `Component`s (rule-based/ml/dl). It allows to train and infer models in a pipeline as a whole.



DeepPavlov installation

- Ubuntu

Create a virtual environment with Python 3.6

```
virtualenv -p python3.6 env
```

Activate the environment.

```
source ./env/bin/activate
```

Clone the repo and cd to project root

```
git clone https://github.com/deepmipt/DeepPavlov.git cd DeepPavlov
```

Install the requirements:

```
python setup.py develop
```

Install spacy dependencies:

```
python -m spacy download en
```

DeepPavlov installation

- Windows

Install the Docker following the instructions:

https://docs.docker.com/docker-for-windows/install

Then go to console and get the container by the following command:

```
docker pull altinsky/convai:deeppavlov
```

Run the container with DeepPavlov installation:

```
docker run -p 8888:8888 altinsky/convai:deeppavlov
```

Open http://127.0.0.1:8888/ in your browser to access Jupyter Notebook

Upload file with tutorial via Jupyter Notebook

To STOP the container:

```
docker stop
```

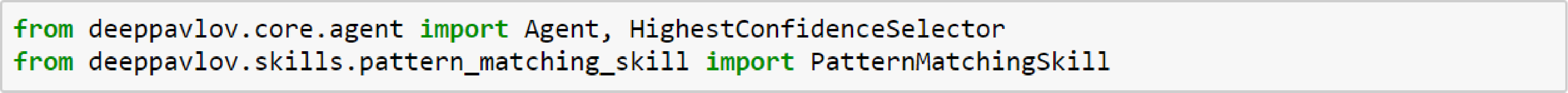
To continue working with your saved container:

`docker ps -a` to list saved containers

```
docker start _your_container_id_
```

HelloBot in 7 lines in DeepPavlov

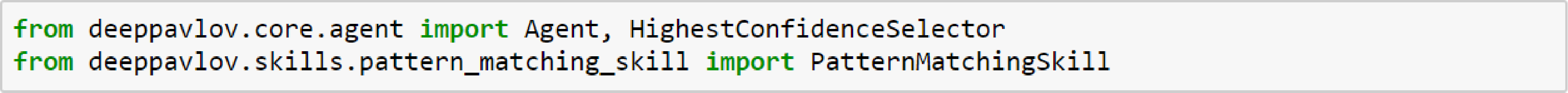
- Import core components of the dialogue `Agent`



```python
from deeppavlov.core.agent import Agent, HighestConfidenceSelector
from deeppavlov.skills.pattern_matching_skill import PatternMatchingSkill
```

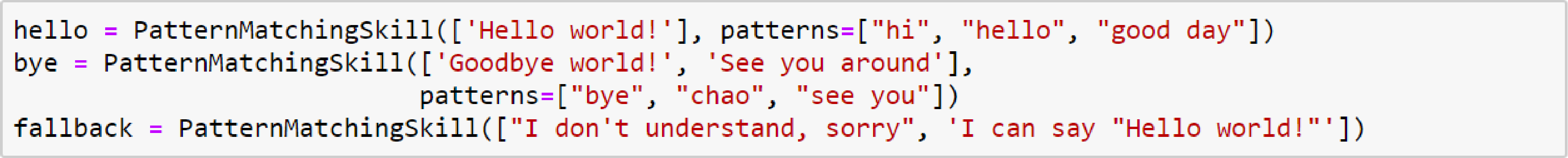
HelloBot in 7 lines in DeepPavlov

- Import core components of the dialogue `Agent`



```
from deeppavlov.core.agent import Agent, HighestConfidenceSelector
from deeppavlov.skills.pattern_matching_skill import PatternMatchingSkill
```

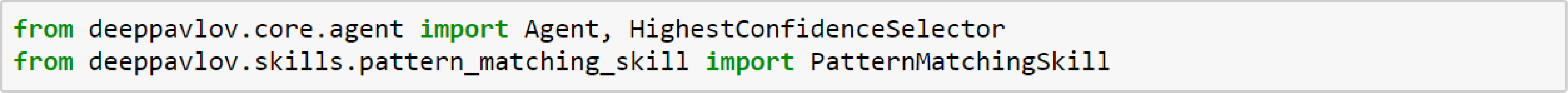
- Define responses and input patterns for `Skills`



```
hello = PatternMatchingSkill(['Hello world!'], patterns=["hi", "hello", "good day"])
bye = PatternMatchingSkill(['Goodbye world!', 'See you around'],
                           patterns=["bye", "chao", "see you"])
fallback = PatternMatchingSkill(["I don't understand, sorry", 'I can say "Hello world!"'])
```

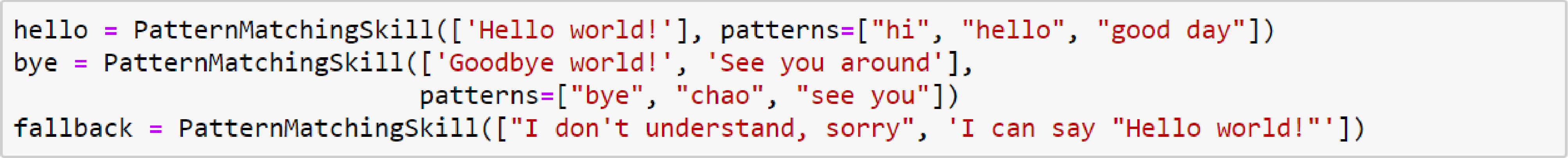
HelloBot in 7 lines in DeepPavlov

- Import core components of the dialogue `Agent`



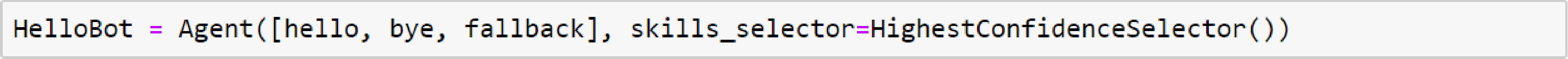
```python
from deeppavlov.core.agent import Agent, HighestConfidenceSelector
from deeppavlov.skills.pattern_matching_skill import PatternMatchingSkill
```

- Define responses and input patterns for `Skills`



```python
hello = PatternMatchingSkill(['Hello world!'], patterns=["hi", "hello", "good day"])
bye = PatternMatchingSkill(['Goodbye world!', 'See you around'],
                           patterns=["bye", "chao", "see you"])
fallback = PatternMatchingSkill(["I don't understand, sorry", 'I can say "Hello world!"'])
```

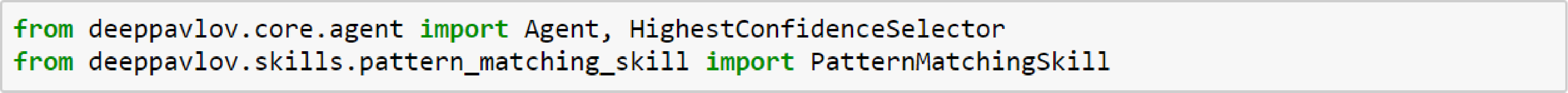
- Combine `Skills` with `SkillManager` (selector) into an `Agent`



```python
HelloBot = Agent([hello, bye, fallback], skills_selector=HighestConfidenceSelector())
```

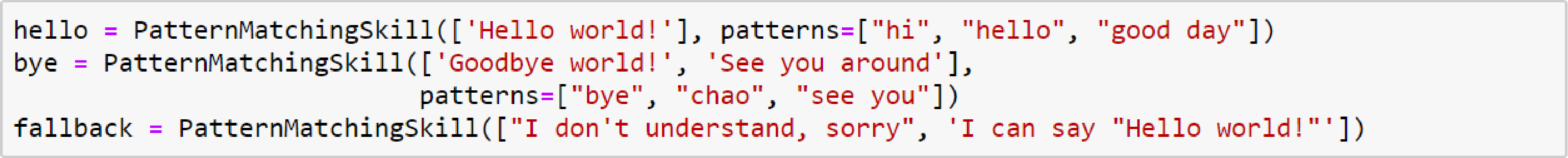
HelloBot in 7 lines in DeepPavlov

- Import core components of the dialogue `Agent`



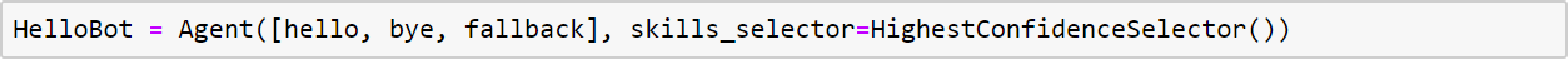
```
from deeppavlov.core.agent import Agent, HighestConfidenceSelector
from deeppavlov.skills.pattern_matching_skill import PatternMatchingSkill
```

- Define responses and input patterns for `Skills`



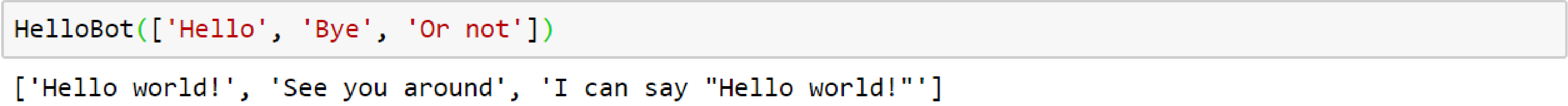
```
hello = PatternMatchingSkill(['Hello world!'], patterns=["hi", "hello", "good day"])
bye = PatternMatchingSkill(['Goodbye world!', 'See you around'],
                           patterns=["bye", "chao", "see you"])
fallback = PatternMatchingSkill(["I don't understand, sorry", 'I can say "Hello world!"'])
```

- Combine `Skills` with `SkillManager` (selector) into an `Agent`



```
HelloBot = Agent([hello, bye, fallback], skills_selector=HighestConfidenceSelector())
```

- Talk with **HelloBot!**



```
HelloBot(['Hello', 'Bye', 'Or not'])
```

```
['Hello world!', 'See you around', 'I can say "Hello world!"']
```

Diving deeper in DeepPavlov

- Simple `skills` are boring!
- Trainable `skills` are cool!

Diving deeper in DeepPavlov

- How to build advanced bot

- How to build advanced bot
  - for every `Skill`
    - prepare data
    - define trainable model
    - train model

- How to build advanced bot
  - for every `Skill`
    - prepare data
    - define trainable model
    - train model
  - create `SkillManager` = `SkillSelector` + `SkillFilter`

- How to build advanced bot
  - for every `Skill`
    - prepare data
    - define trainable model
    - train model
  - create `SkillManager` = `SkillSelector` + `SkillFilter`
  - assemble `Skills` and `SkillManager` into an `Agent`

- How to build advanced bot
  - for every `Skill`
    - prepare data
    - define trainable model
    - train model
  - create `SkillManager` = `SkillSelector` + `SkillFilter`
  - assemble `Skills` and `SkillManager` into an `Agent`

- Data preparation
  - Read data – `DatasetReader`
  - Index data – `Vocab`
  - Manage Data - `DatasetIterator`

- How to build advanced bot
  - for every `Skill`
    - prepare data
    - define trainable model
    - train model
  - create `SkillManager` = `SkillSelector` + `SkillFilter`
  - assemble `Skills` and `SkillManager` into an `Agent`

Features

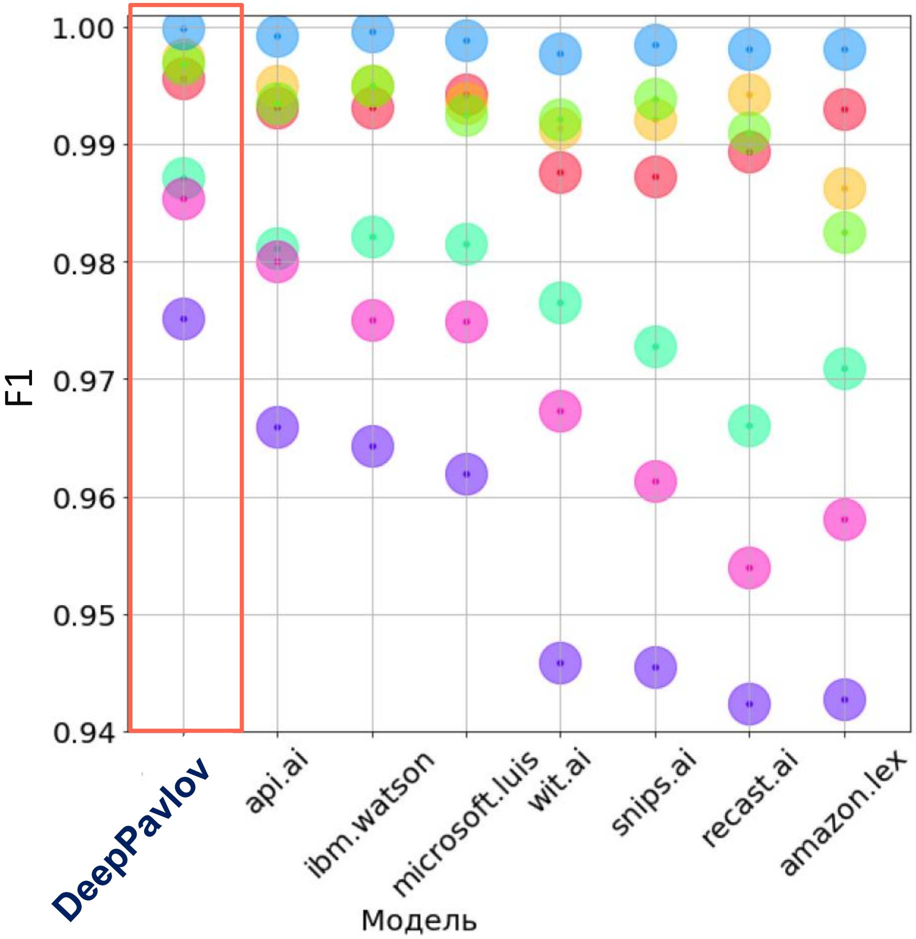
Component	Description
NER component	Based on neural Named Entity Recognition network. The NER component reproduces architecture from the paper Application of a Hybrid Bi-LSTM-CRF model to the task of Russian Named Entity Recognitionwhich is inspired by Bi-LSTM+CRF architecture from https://arxiv.org/pdf/1603.01360.pdf.
Slot filling components	Based on fuzzy Levenshtein search to extract normalized slot values from text. The components either rely on NER results or perform needle in haystack search.
Classification component	Component for classification tasks (intents, sentiment, etc). Based on shallow-and-wide Convolutional Neural Network architecture from Kim Y. Convolutional neural networks for sentence classification – 2014 and others. The model allows multilabel classification of sentences.
Automatic spelling correction component	Pipelines that use candidates search in a static dictionary and an ARPA language model to correct spelling errors.
Ranking component	Based on LSTM-based deep learning models for non-factoid answer selection. The model performs ranking of responses or contexts from some database by their relevance for the given context.
Question Answering component	Based on R-NET: Machine Reading Comprehension with Self-matching Networks. The model solves the task of looking for an answer on a question in a given context (SQuAD task format).
Morphological tagging component	Based on character-based approach to morphological tagging Heigold et al., 2017. An extensive empirical evaluation of character-based morphological tagging for 14 languages. A state-of-the-art model for Russian and several other languages. Model assigns morphological tags in UD format to sequences of words.
Skills	
Goal-oriented bot	Based on Hybrid Code Networks (HCNs) architecture from Jason D. Williams, Kavosh Asadi, Geoffrey Zweig, Hybrid Code Networks: practical and efficient end-to-end dialog control with supervised and reinforcement learning – 2017. It allows to predict responses in goal-oriented dialog. The model is customizable: embeddings, slot filler and intent classifier can switched on and off on demand.
Seq2seq goal-oriented bot	Dialogue agent predicts responses in a goal-oriented dialog and is able to handle multiple domains (pretrained bot allows calendar scheduling, weather information retrieval, and point-of-interest navigation). The model is end-to-end differentiable and does not need to explicitly model dialogue state or belief trackers.
ODQA	An open domain question answering skill. The skill accepts free-form questions about the world and outputs an answer based on its Wikipedia knowledge.
Embeddings	
Pre-trained embeddings for the Russian language	Word vectors for the Russian language trained on joint Russian Wikipedia and Lenta.ru corpora.

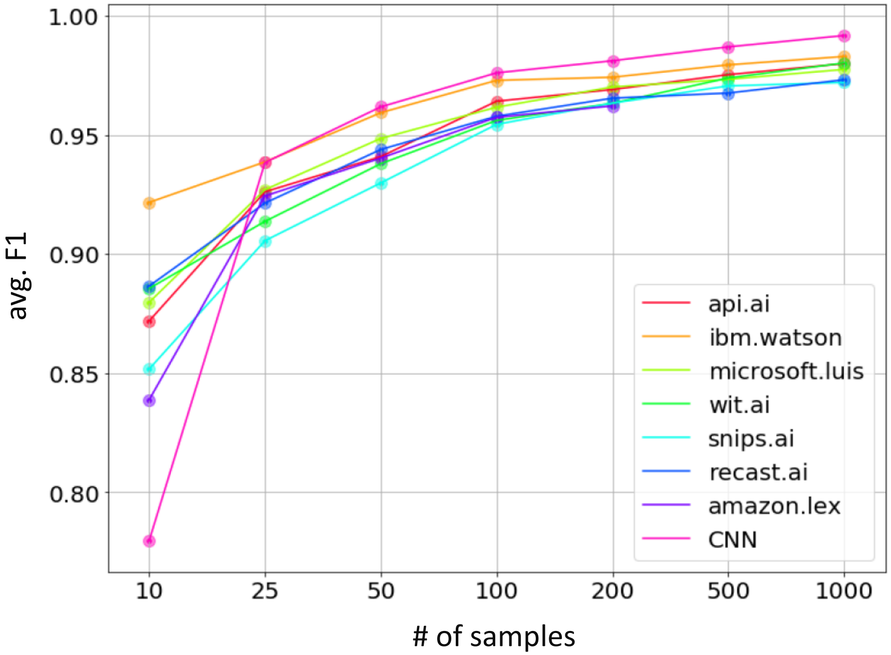
Some results

- Intent recognition

Yoon Kim. 2014. Convolutional neural networks for sentence classification.

	F1-score
DeepPavlov	**99.10**
api.ai[6]	98.68
IBM Watson[7]	98.63
Microsoft LUIS[8]	98.53
Wit.ai[9]	97.97
Snips.ai[10]	97.87
Recast.ai[11]	97.64
Amazon Lex[12]	97.59





iPavlov.ai

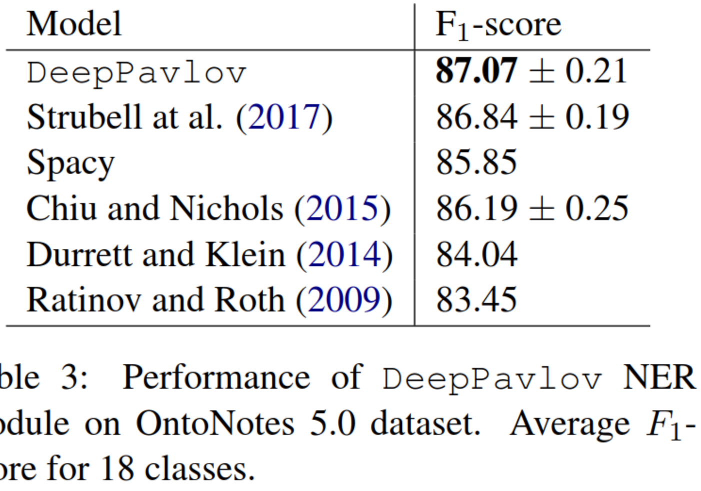
Source of all data except DeepPavlov is https://www.slideshare.net/KonstantinSavenkov/nlu-intent-detection-benchmark-by-intento-august-2017

MIPT MOSCOW INSTITUTE OF PHYSICS AND TECHNOLOGY

Some results

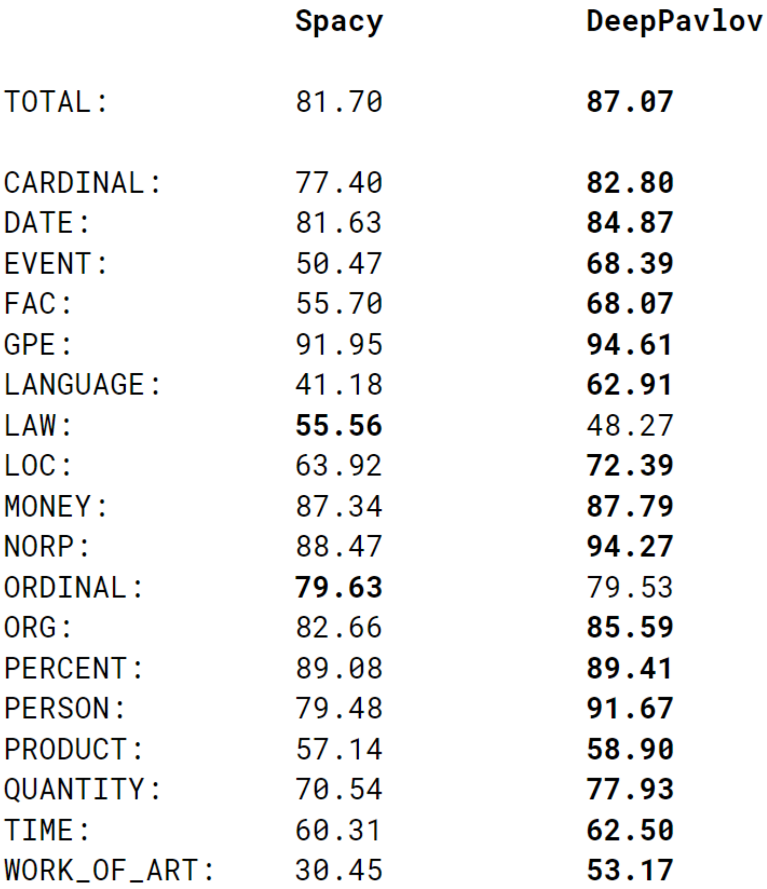
- Entity recognition

Le Tanh Anh, Mikhail Y Arkhipov, and Mikhail S Burtsev. 2017. Application of a hybrid bi-lstm-crf model to the task of russian named entity recognition.



Model	F_1-score
DeepPavlov	**87.07** ± 0.21
Strubell at al. (2017)	86.84 ± 0.19
Spacy	85.85
Chiu and Nichols (2015)	86.19 ± 0.25
Durrett and Klein (2014)	84.04
Ratinov and Roth (2009)	83.45

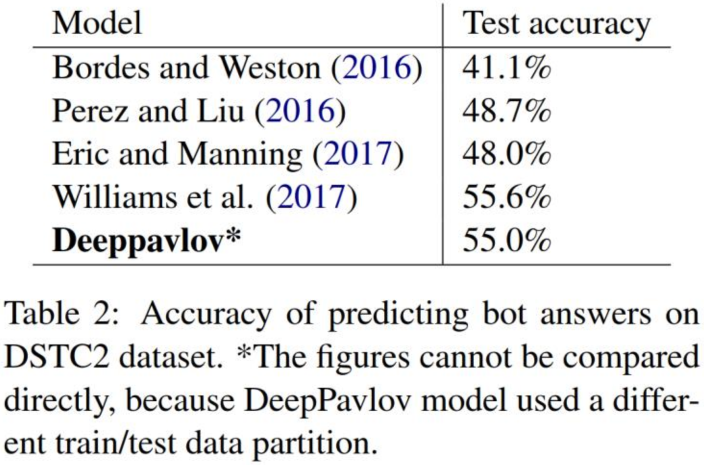
Table 3: Performance of DeepPavlov NER module on OntoNotes 5.0 dataset. Average F_1-score for 18 classes.



	Spacy	DeepPavlov
TOTAL:	81.70	**87.07**
CARDINAL:	77.40	**82.80**
DATE:	81.63	**84.87**
EVENT:	50.47	**68.39**
FAC:	55.70	**68.07**
GPE:	91.95	**94.61**
LANGUAGE:	41.18	**62.91**
LAW:	**55.56**	48.27
LOC:	63.92	**72.39**
MONEY:	87.34	**87.79**
NORP:	88.47	**94.27**
ORDINAL:	**79.63**	79.53
ORG:	82.66	**85.59**
PERCENT:	89.08	**89.41**
PERSON:	79.48	**91.67**
PRODUCT:	57.14	**58.90**
QUANTITY:	70.54	**77.93**
TIME:	60.31	**62.50**
WORK_OF_ART:	30.45	**53.17**

- Goal-oriented dialogue

Jason D Williams, Kavosh Asadi, and Geoffrey Zweig. 2017. Hybrid code networks: practical and efficient end-to-end dialog control with supervised and reinforcement learning.



Model	Test accuracy
Bordes and Weston (2016)	41.1%
Perez and Liu (2016)	48.7%
Eric and Manning (2017)	48.0%
Williams et al. (2017)	55.6%
Deeppavlov*	55.0%

Table 2: Accuracy of predicting bot answers on DSTC2 dataset. *The figures cannot be compared directly, because DeepPavlov model used a different train/test data partition.

iPavlov.ai

```
# Definition of iPavlov project
def iPavlov(talent, ideas):
   research = ideas * talent
   AI = development(research)
   return AI
# How you are related to the iPavlov project
email.send(‘merge@ipavlov.ai’, YOU.CV)
if YOU in [‘researcher’,
           ‘developer’]
       and YOU is (‘ai_geek’ &
                   ‘performer’ &
                   ‘team_player’):
       iPavlov(YOU.talent, YOU.ideas)
```

Interactive demo

http://demo.ipavlov.ai/

Source code

https://github.com/deepmipt/