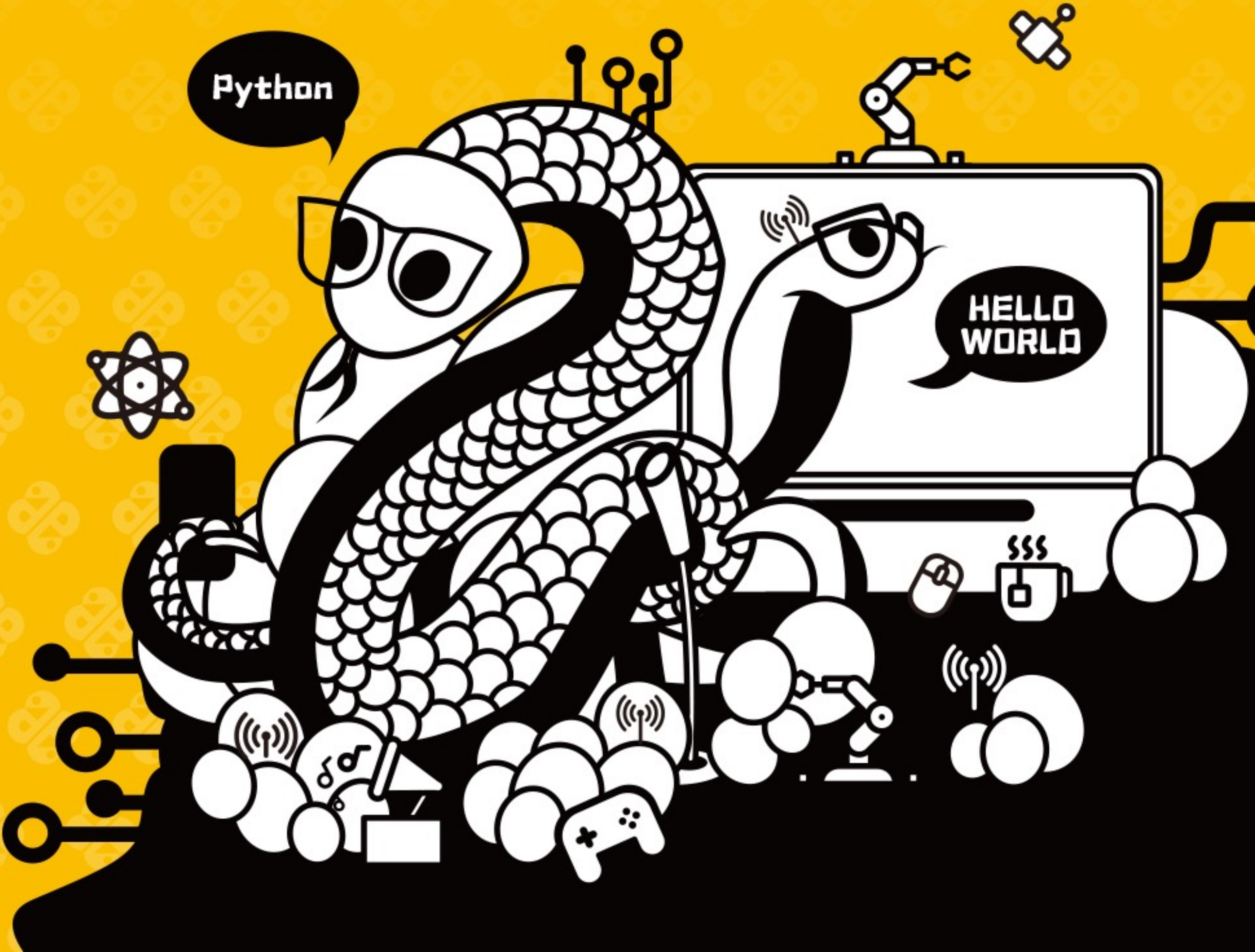
python™ × PYCHINA Python 中国社区

Python for Good

PyCon China 2022

基于海量数据的实体关系抽取实践

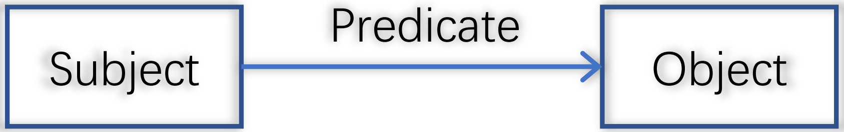
主讲人： 杨 华 – 海豚跃跃联合创始人



应用场景

- ➢ 知识图谱构建的核心关键

- ➢ SPO三元组

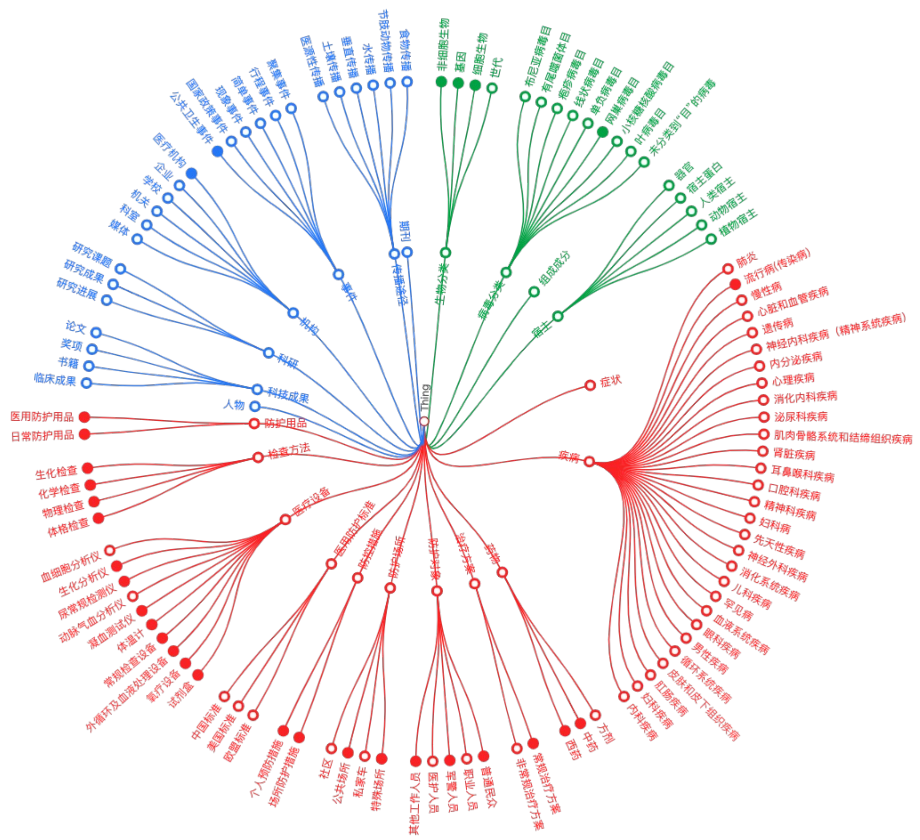


Subject

Predicate

Object

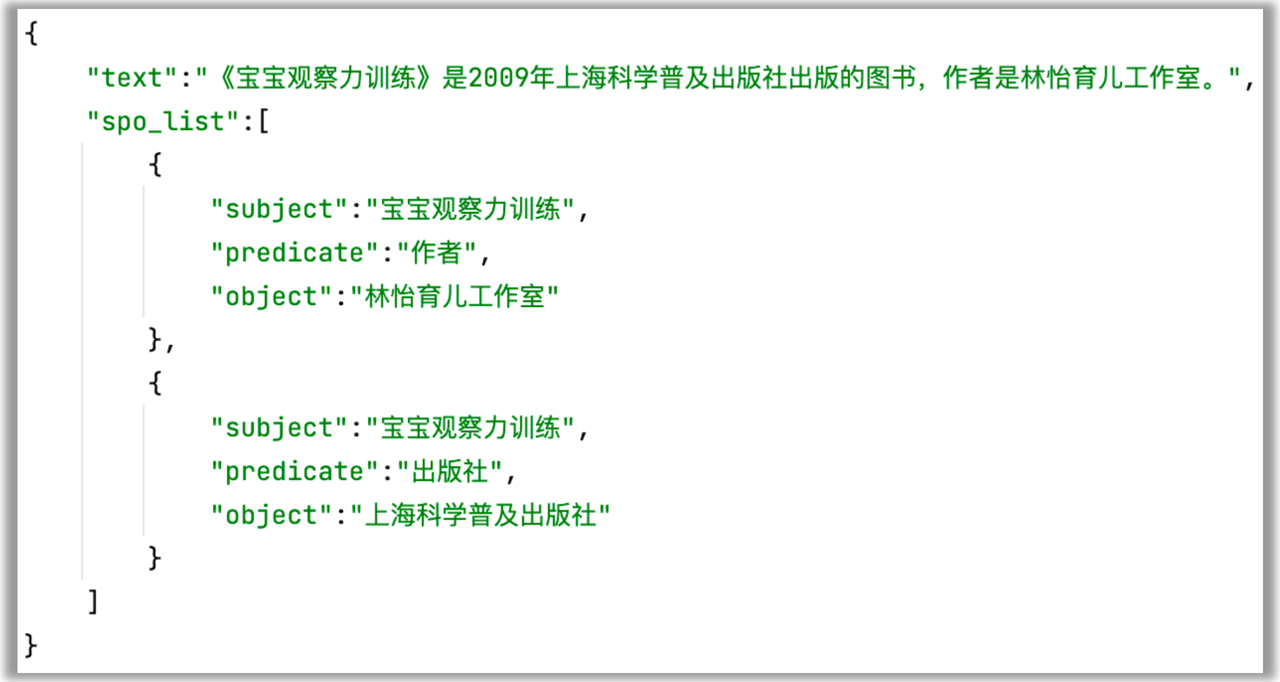
- ➢ 基于Bert预训练模型的实体关系抽取实践



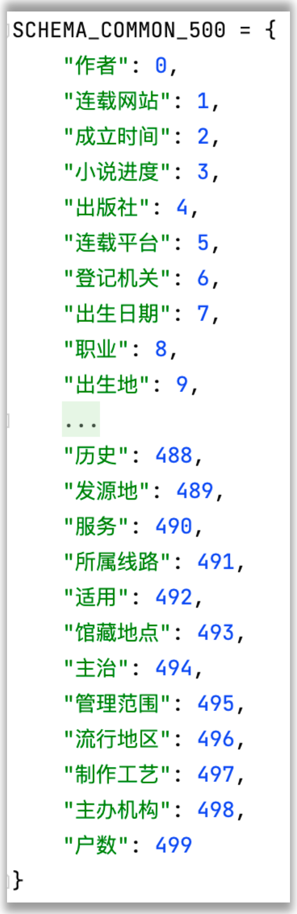
COKG-19 知识图谱

数据与网络结构：训练数据

- 7 Mio Sample
- 15 Mio SPO
- 500 Schema



```
{
    "text":"《宝宝观察力训练》是2009年上海科学普及出版社出版的图书，作者是林怡育儿工作室。",
    "spo_list":[
        {
            "subject":"宝宝观察力训练",
            "predicate":"作者",
            "object":"林怡育儿工作室"
        },
        {
            "subject":"宝宝观察力训练",
            "predicate":"出版社",
            "object":"上海科学普及出版社"
        }
    ]
}
```

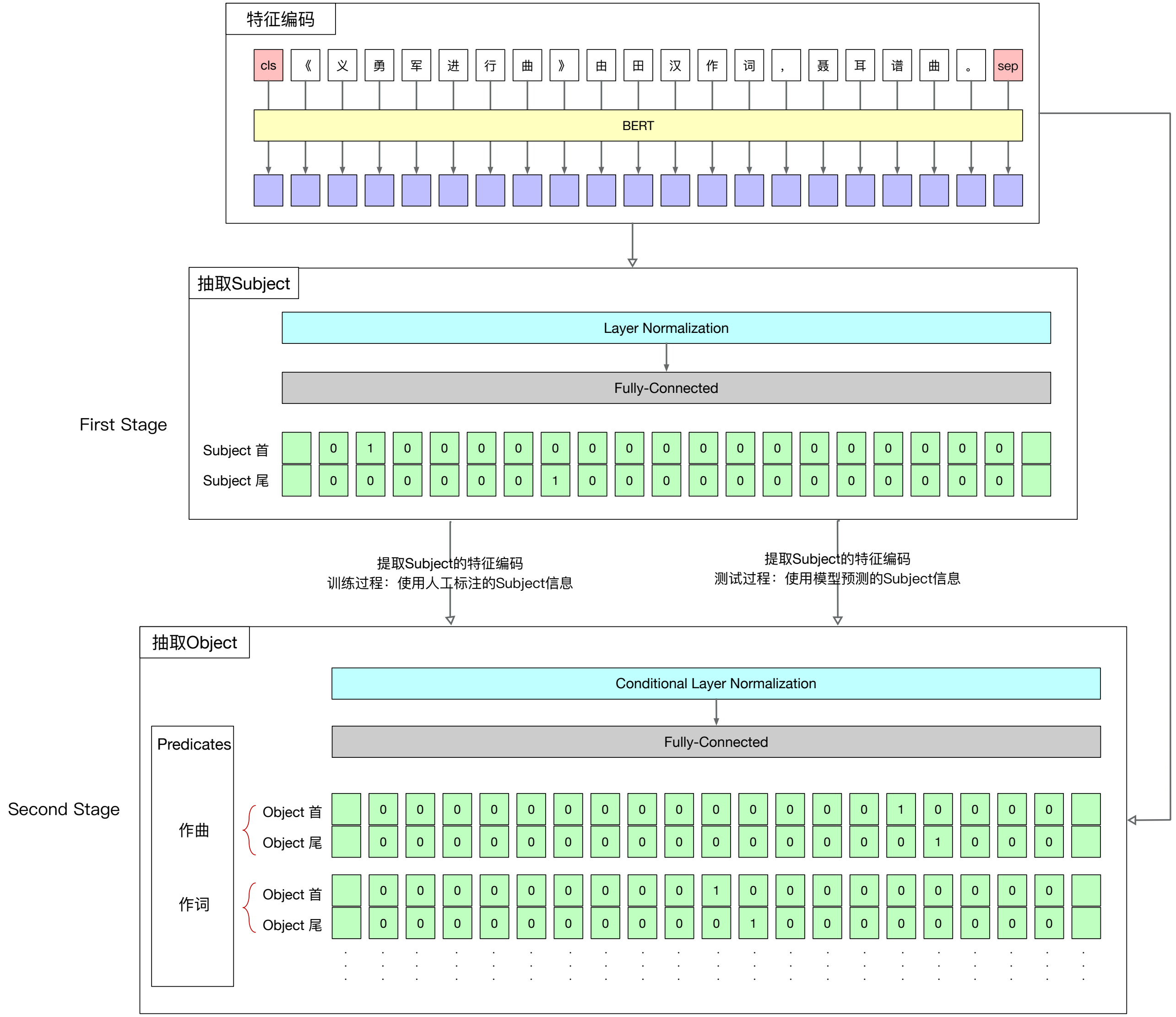


```
SCHEMA_COMMON_500 = {
    "作者": 0,
    "连载网站": 1,
    "成立时间": 2,
    "小说进度": 3,
    "出版社": 4,
    "连载平台": 5,
    "登记机关": 6,
    "出生日期": 7,
    "职业": 8,
    "出生地": 9,
    ...
    "历史": 488,
    "发源地": 489,
    "服务": 490,
    "所属线路": 491,
    "适用": 492,
    "馆藏地点": 493,
    "主治": 494,
    "管理范围": 495,
    "流行地区": 496,
    "制作工艺": 497,
    "主办机构": 498,
    "户数": 499
}
```

数据与网络结构：Joint Extraction

- ➢ 两阶段的实体关系抽取网络
  - ✓ 两分类任务
    - 一阶段识别Subject
    - 二阶段识别Object
  - ✓ 无法解决实体嵌套： “上海迪士尼”

	上	海	迪	士	尼
Subject 首	1	0	1	0	0
Subject 尾	0	1	0	0	1



提取Subject的特征编码
训练过程：使用人工标注的Subject信息

提取Subject的特征编码
测试过程：使用模型预测的Subject信息

	cls	《	义	勇	军	进	行	曲	》	由	田	汉	作	词	，	聂	耳	谱	曲	。	sep
Subject 首		0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Subject 尾		0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	

		cls	《	义	勇	军	进	行	曲	》	由	田	汉	作	词	，	聂	耳	谱	曲	。	sep
作曲	Object 首		0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	
	Object 尾		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	
作词	Object 首		0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	
	Object 尾		0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	

数据与网络结构：Global Pointer

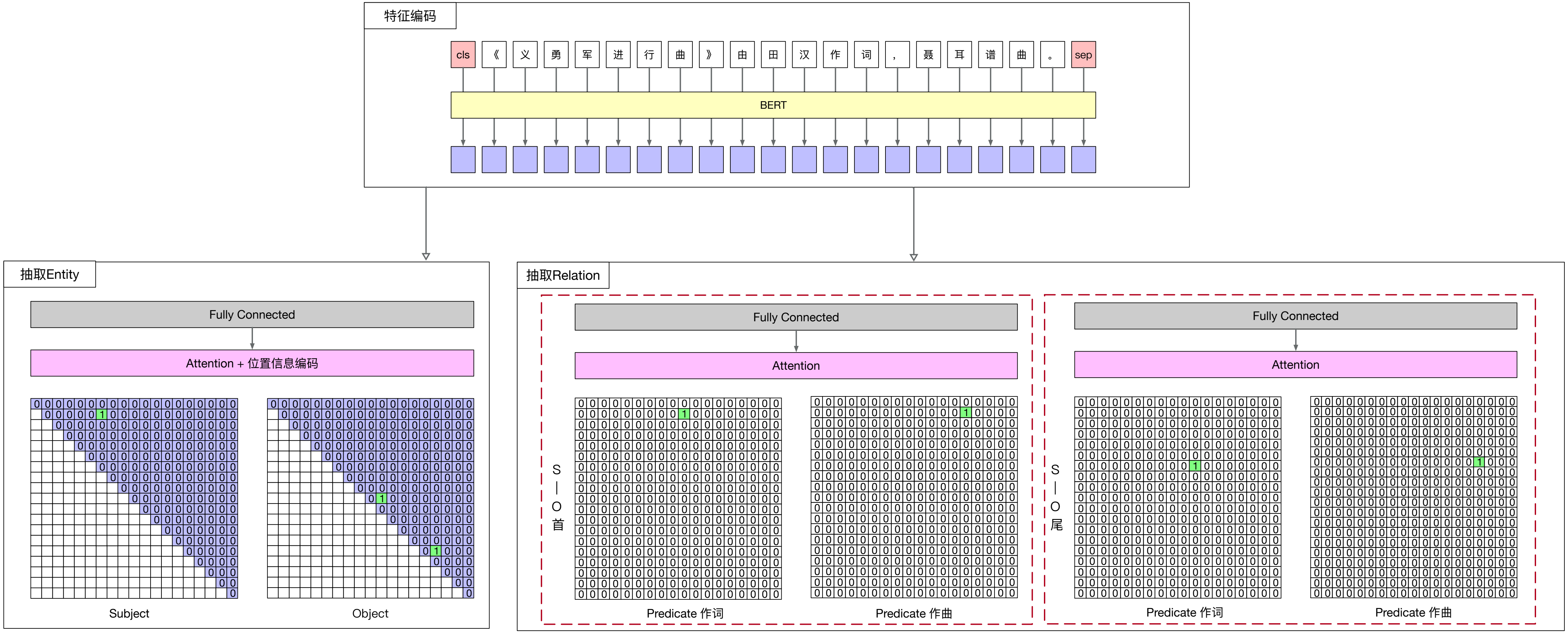
- 全局归一化的实体关系抽取网络
  - ✓ 升维解决实体嵌套

	上	海	迪	士	尼
上	0	1	0	0	1
海		0	0	0	0
迪			0	0	1
士				0	0
尼					0

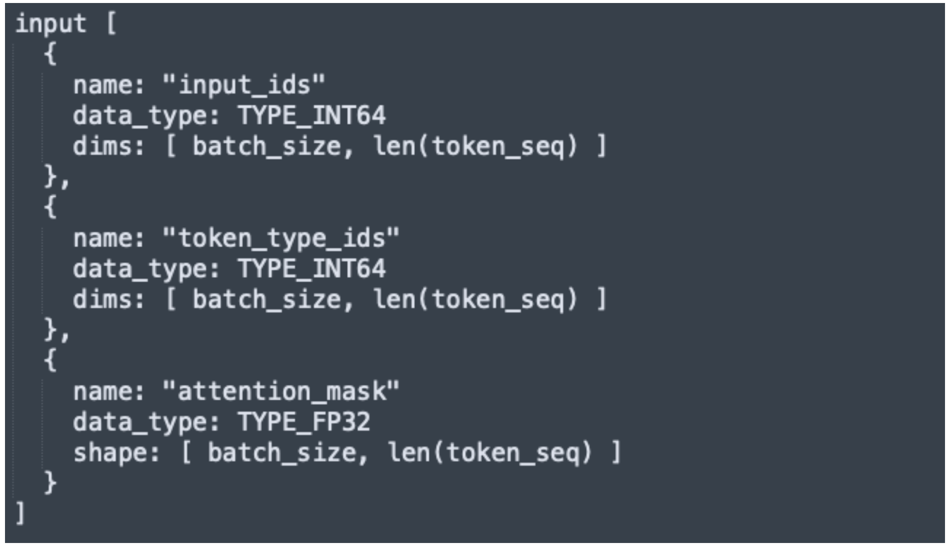
	cls	《	义	勇	军	进	行	曲	》	由	田	汉	作	词	，	聂	耳	谱	曲	。	sep
cls																					
《		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
义			0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	
勇				0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
军					0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
进						0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
行							0	0	0	0	0	0	0	0	0	0	0	0	0	0	
曲								0	0	0	0	0	0	0	0	0	0	0	0	0	
》									0	0	0	0	0	0	0	0	0	0	0	0	
由										0	0	0	0	0	0	0	0	0	0	0	
田											0	0	0	0	0	0	0	0	0	0	
汉												0	0	0	0	0	0	0	0	0	
作													0	0	0	0	0	0	0	0	
词														0	0	0	0	0	0	0	
，															0	0	0	0	0	0	
聂																0	0	0	0	0	
耳																	0	0	0	0	
谱																		0	0	0	
曲																			0	0	
。																				0	
sep																					

数据与网络结构：Global Pointer

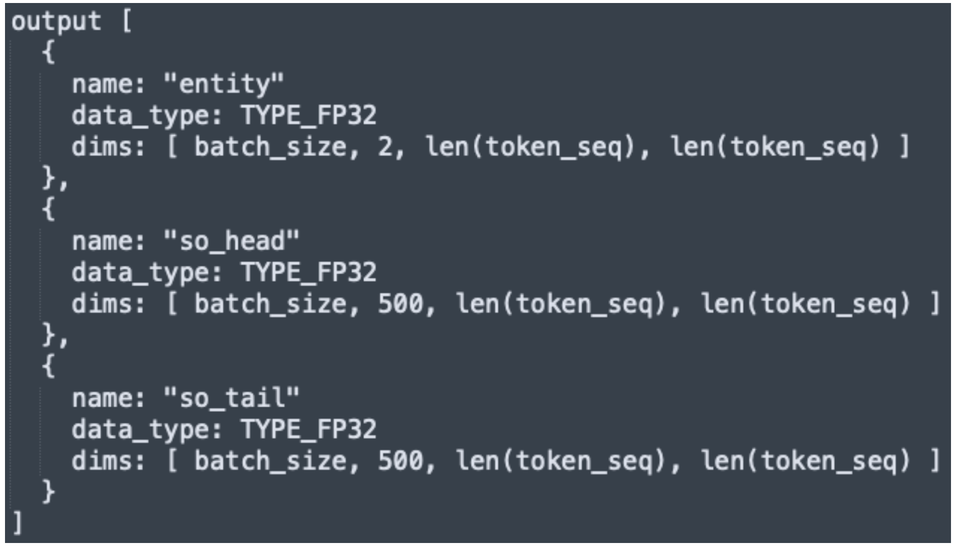
- 全局归一化的实体关系抽取网络



数据与网络结构：模型输入、输出



```
input [
  {
    name: "input_ids"
    data_type: TYPE_INT64
    dims: [ batch_size, len(token_seq) ]
  },
  {
    name: "token_type_ids"
    data_type: TYPE_INT64
    dims: [ batch_size, len(token_seq) ]
  },
  {
    name: "attention_mask"
    data_type: TYPE_FP32
    shape: [ batch_size, len(token_seq) ]
  }
]
```



```
output [
  {
    name: "entity"
    data_type: TYPE_FP32
    dims: [ batch_size, 2, len(token_seq), len(token_seq) ]
  },
  {
    name: "so_head"
    data_type: TYPE_FP32
    dims: [ batch_size, 500, len(token_seq), len(token_seq) ]
  },
  {
    name: "so_tail"
    data_type: TYPE_FP32
    dims: [ batch_size, 500, len(token_seq), len(token_seq) ]
  }
]
```

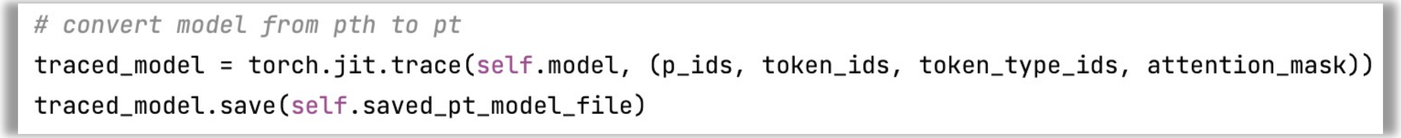
Training Environment	nvcr.io/nvidia/pytorch:22.05-py3
GPU	NVIDIA A100 (× 4)
Pretrain Model	RoBERTa-wwm-ext-large, Chinese

Max_seq_len	256
Batch Size	24
Initialization	Kaiming normal initialization
Optimizer	Adam with betas(0.9, 0.999) L2 penalty 1e-2
Loss	Sparse multi-label cross-entropy
Initial learning Rate	1e-5
Learning rate decay schedule	Step decay with warmup

测试指标	F1=80.84%

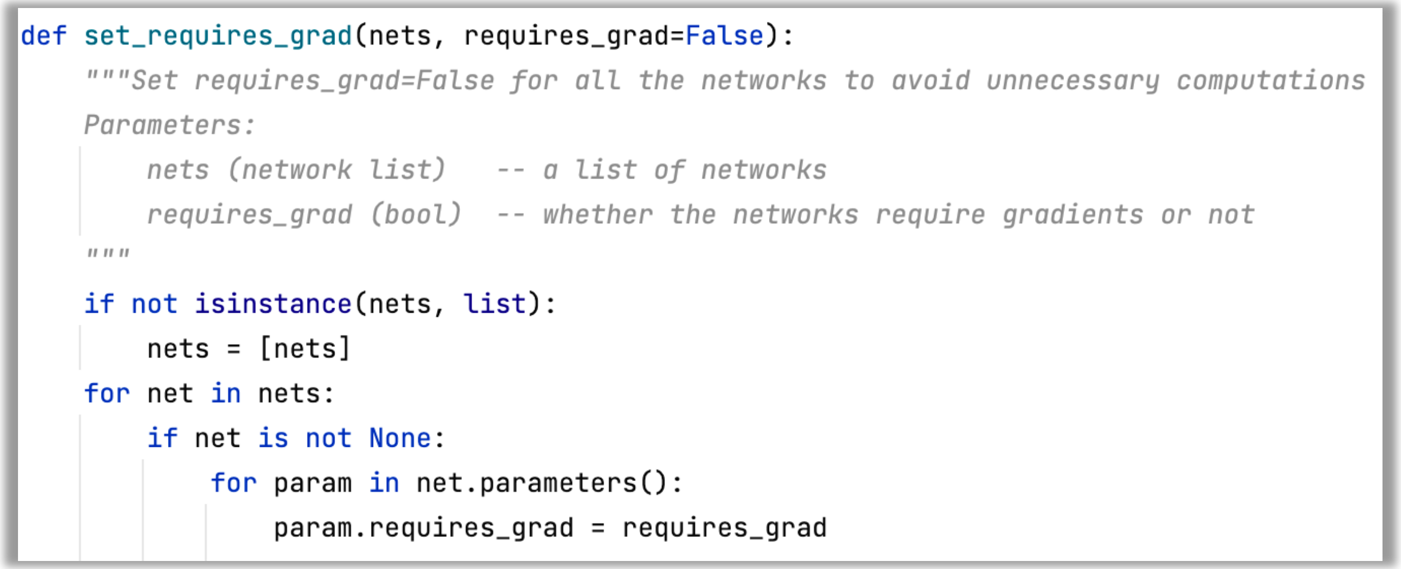
基于Nvidia Triton的模型部署--TorchScript

Step 1--导出TorchScript模型

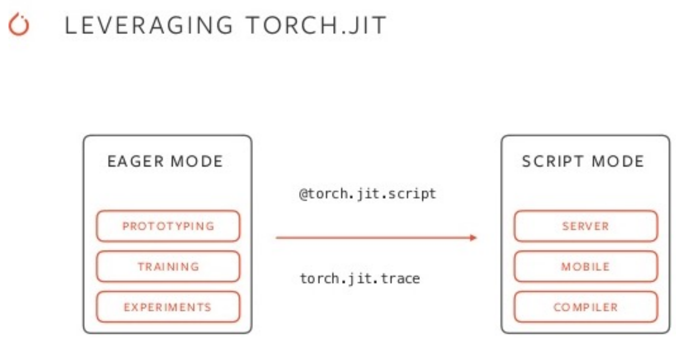


```
# convert model from pth to pt
traced_model = torch.jit.trace(self.model, (p_ids, token_ids, token_type_ids, attention_mask))
traced_model.save(self.saved_pt_model_file)
```

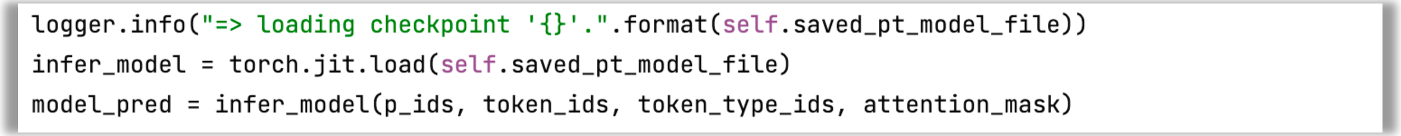
注意：转换前需要把模型参数的requires_grad设置为false，否则显存占用会飙升。



```
def set_requires_grad(nets, requires_grad=False):
    """Set requires_grad=False for all the networks to avoid unnecessary computations
    Parameters:
        nets (network list)   -- a list of networks
        requires_grad (bool)  -- whether the networks require gradients or not
    """
    if not isinstance(nets, list):
        nets = [nets]
    for net in nets:
        if net is not None:
            for param in net.parameters():
                param.requires_grad = requires_grad
```



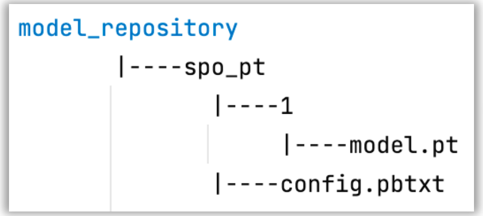
Step 2--加载TorchScript模型，完成一致性校验



```
logger.info("=> loading checkpoint '{}'.".format(self.saved_pt_model_file))
infer_model = torch.jit.load(self.saved_pt_model_file)
model_pred = infer_model(p_ids, token_ids, token_type_ids, attention_mask)
```

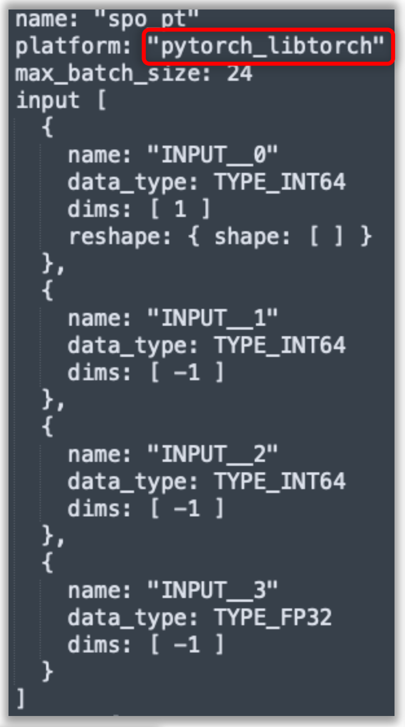
基于Nvidia Triton的模型部署--TorchScript

Step 3--创建 model repository

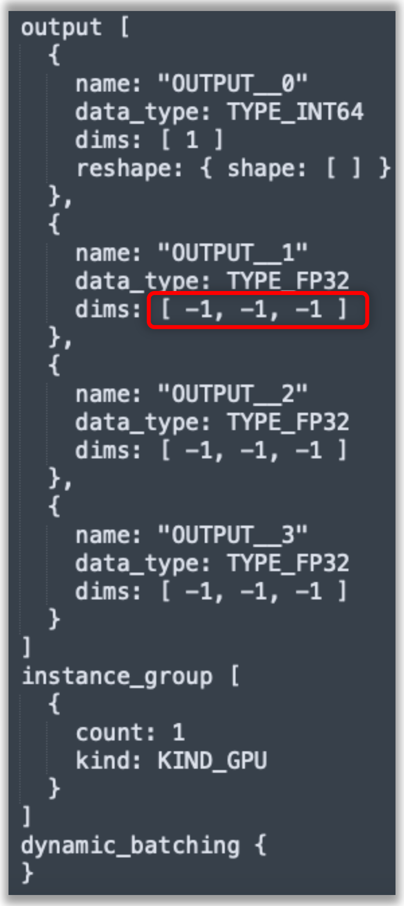


```
model_repository
        |----spo_pt
                |----1
                     |----model.pt
                |----config.pbtxt
```

config.pbtxt

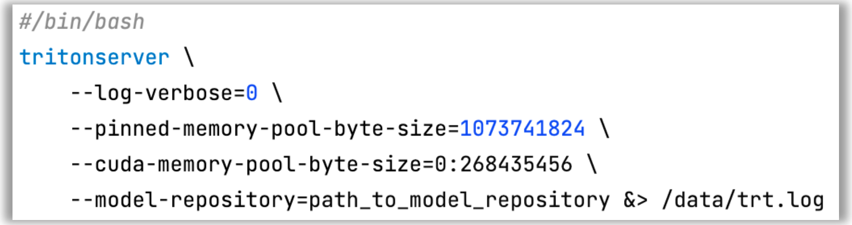


```
name: "spo_pt"
platform: "pytorch_libtorch"
max_batch_size: 24
input [
  {
    name: "INPUT__0"
    data_type: TYPE_INT64
    dims: [ 1 ]
    reshape: { shape: [ ] }
  },
  {
    name: "INPUT__1"
    data_type: TYPE_INT64
    dims: [ -1 ]
  },
  {
    name: "INPUT__2"
    data_type: TYPE_INT64
    dims: [ -1 ]
  },
  {
    name: "INPUT__3"
    data_type: TYPE_FP32
    dims: [ -1 ]
  }
]
```



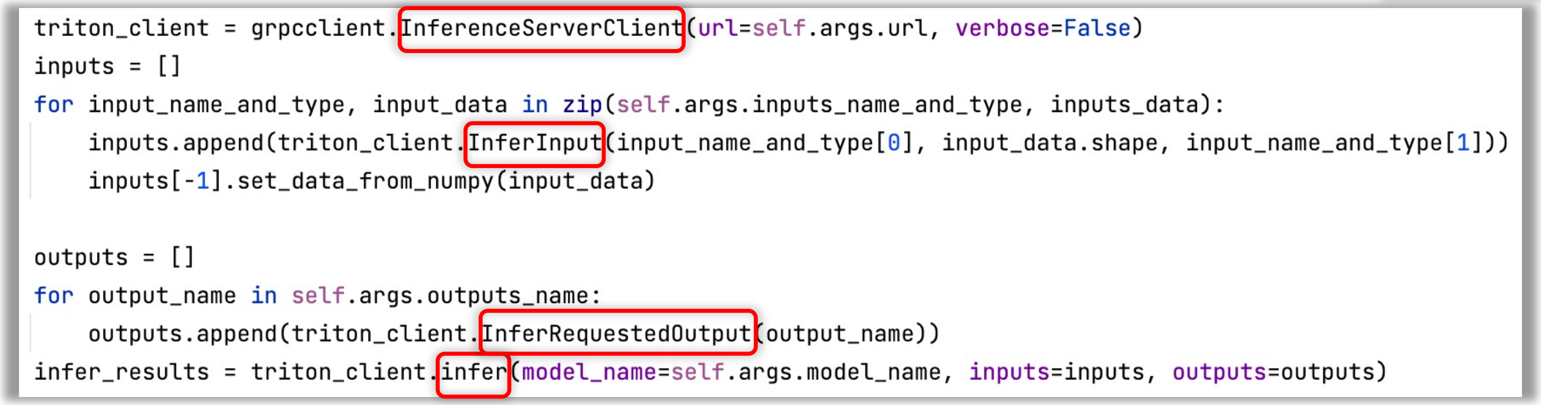
```
output [
  {
    name: "OUTPUT__0"
    data_type: TYPE_INT64
    dims: [ 1 ]
    reshape: { shape: [ ] }
  },
  {
    name: "OUTPUT__1"
    data_type: TYPE_FP32
    dims: [ -1, -1, -1 ]
  },
  {
    name: "OUTPUT__2"
    data_type: TYPE_FP32
    dims: [ -1, -1, -1 ]
  },
  {
    name: "OUTPUT__3"
    data_type: TYPE_FP32
    dims: [ -1, -1, -1 ]
  }
]
instance_group [
  {
    count: 1
    kind: KIND_GPU
  }
]
dynamic_batching {
}
```

Step 4--Run Triton with GPUs（基础镜像 nvcr.io/nvidia/tritonserver:22.04-py3）



```bash
#/bin/bash
tritonserver \
    --log-verbose=0 \
    --pinned-memory-pool-byte-size=1073741824 \
    --cuda-memory-pool-byte-size=0:268435456 \
    --model-repository=path_to_model_repository &> /data/trt.log
```

Step 5--Run Infer

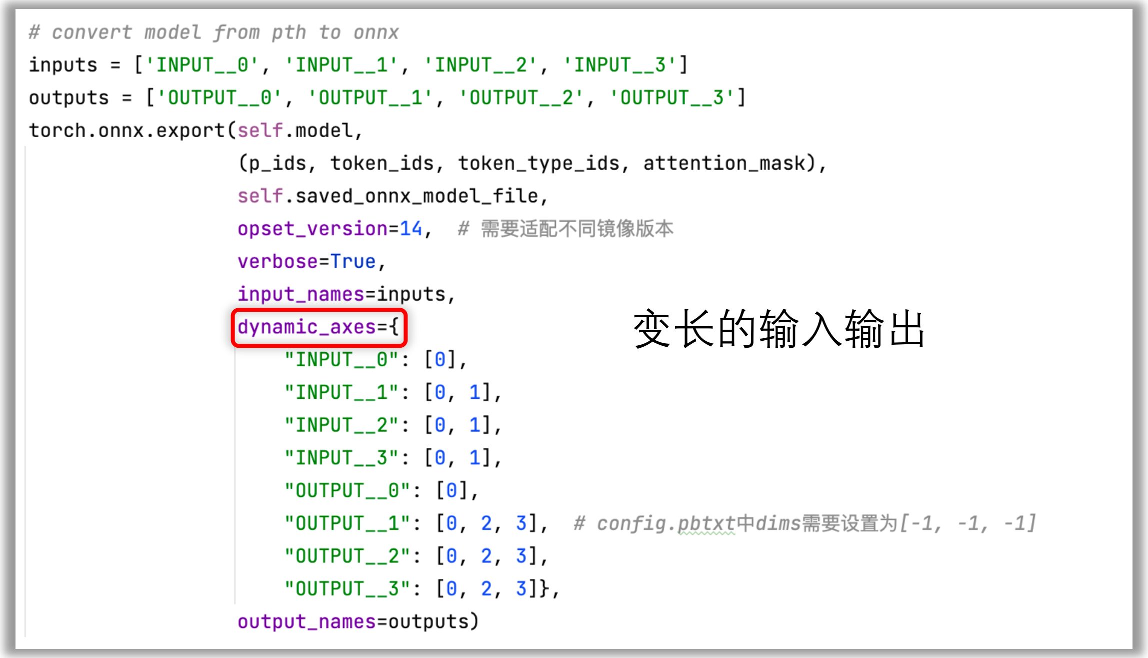


```python
triton_client = grpcclient.InferenceServerClient(url=self.args.url, verbose=False)
inputs = []
for input_name_and_type, input_data in zip(self.args.inputs_name_and_type, inputs_data):
    inputs.append(triton_client.InferInput(input_name_and_type[0], input_data.shape, input_name_and_type[1]))
    inputs[-1].set_data_from_numpy(input_data)

outputs = []
for output_name in self.args.outputs_name:
    outputs.append(triton_client.InferRequestedOutput(output_name))
infer_results = triton_client.infer(model_name=self.args.model_name, inputs=inputs, outputs=outputs)
```

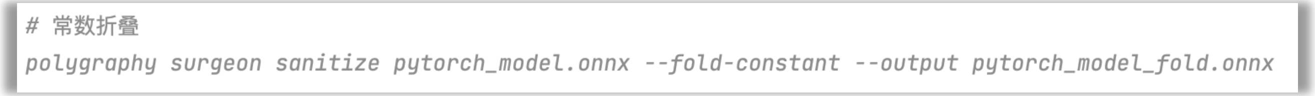
基于Nvidia Triton的模型部署--ONNX

Step 1--导出ONNX模型



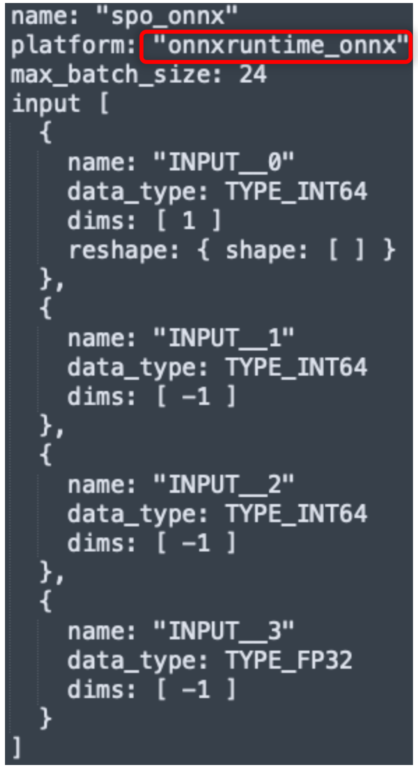
```
# convert model from pth to onnx
inputs = ['INPUT__0', 'INPUT__1', 'INPUT__2', 'INPUT__3']
outputs = ['OUTPUT__0', 'OUTPUT__1', 'OUTPUT__2', 'OUTPUT__3']
torch.onnx.export(self.model,
                  (p_ids, token_ids, token_type_ids, attention_mask),
                  self.saved_onnx_model_file,
                  opset_version=14,  # 需要适配不同镜像版本
                  verbose=True,
                  input_names=inputs,
                  dynamic_axes={
                      "INPUT__0": [0],
                      "INPUT__1": [0, 1],
                      "INPUT__2": [0, 1],
                      "INPUT__3": [0, 1],
                      "OUTPUT__0": [0],
                      "OUTPUT__1": [0, 2, 3],  # config.pbtxt中dims需要设置为[-1, -1, -1]
                      "OUTPUT__2": [0, 2, 3],
                      "OUTPUT__3": [0, 2, 3]},
                  output_names=outputs)
```

变长的输入输出

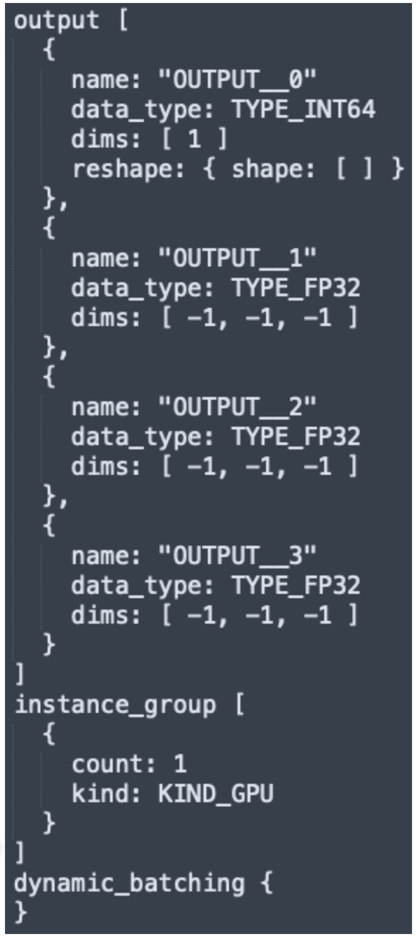


```
# 常数折叠
polygraphy surgeon sanitize pytorch_model.onnx --fold-constant --output pytorch_model_fold.onnx
```

config.pbtxt

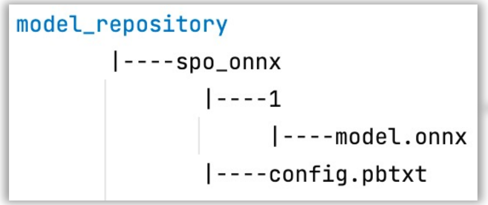


```
name: "spo_onnx"
platform: "onnxruntime_onnx"
max_batch_size: 24
input [
  {
    name: "INPUT__0"
    data_type: TYPE_INT64
    dims: [ 1 ]
    reshape: { shape: [ ] }
  },
  {
    name: "INPUT__1"
    data_type: TYPE_INT64
    dims: [ -1 ]
  },
  {
    name: "INPUT__2"
    data_type: TYPE_INT64
    dims: [ -1 ]
  },
  {
    name: "INPUT__3"
    data_type: TYPE_FP32
    dims: [ -1 ]
  }
]
```



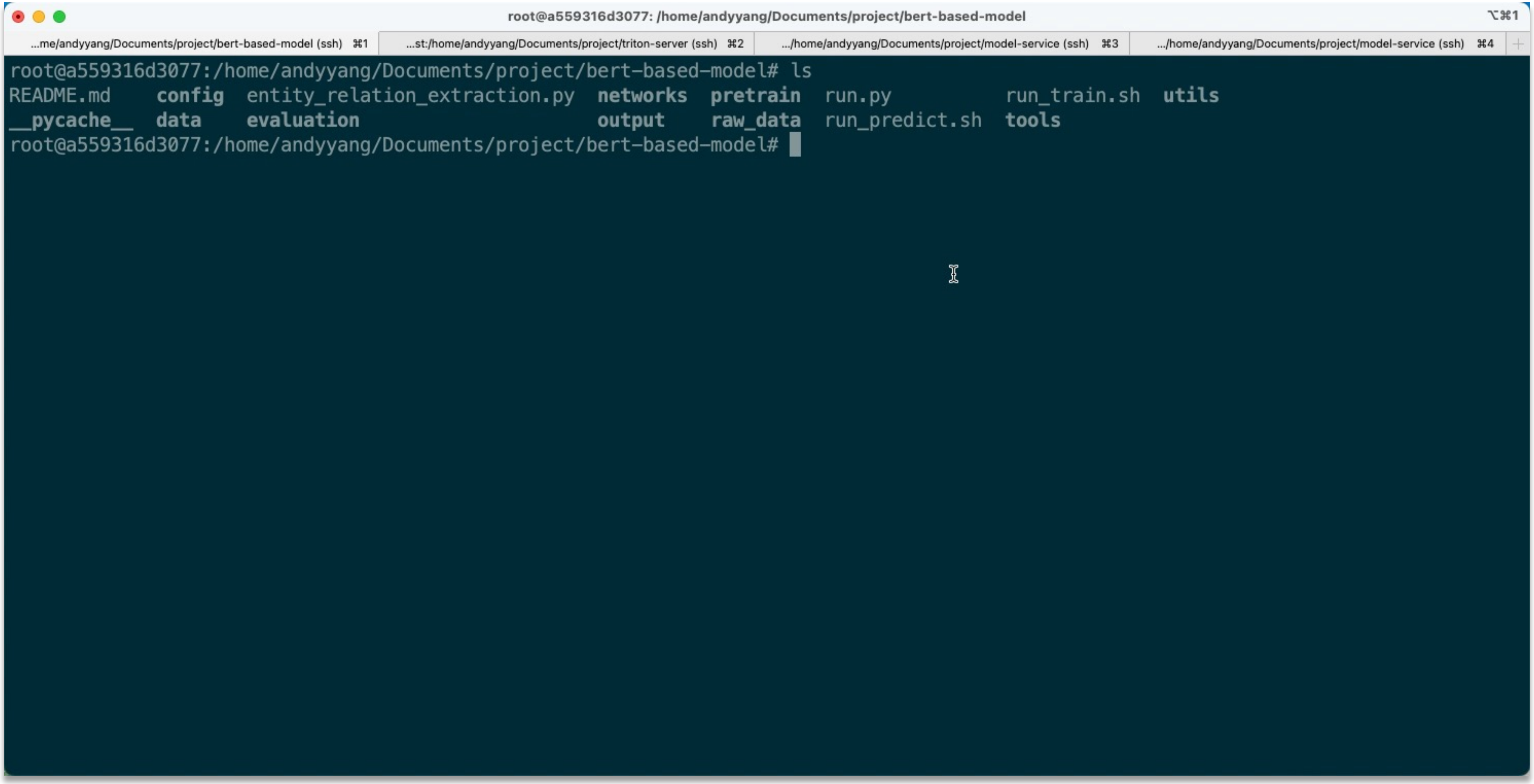
```
output [
  {
    name: "OUTPUT__0"
    data_type: TYPE_INT64
    dims: [ 1 ]
    reshape: { shape: [ ] }
  },
  {
    name: "OUTPUT__1"
    data_type: TYPE_FP32
    dims: [ -1, -1, -1 ]
  },
  {
    name: "OUTPUT__2"
    data_type: TYPE_FP32
    dims: [ -1, -1, -1 ]
  },
  {
    name: "OUTPUT__3"
    data_type: TYPE_FP32
    dims: [ -1, -1, -1 ]
  }
]
instance_group [
  {
    count: 1
    kind: KIND_GPU
  }
]
dynamic_batching {
}
```

Step 3--创建 model repository



```
model_repository
        |----spo_onnx
                |----1
                        |----model.onnx
                |----config.pbtxt
```

效果演示


root@a559316d3077: /home/andyyang/Documents/project/bert-based-model
⌥⌘1
...me/andyyang/Documents/project/bert-based-model (ssh) ⌘1
...st:/home/andyyang/Documents/project/triton-server (ssh) ⌘2
.../home/andyyang/Documents/project/model-service (ssh) ⌘3
.../home/andyyang/Documents/project/model-service (ssh) ⌘4

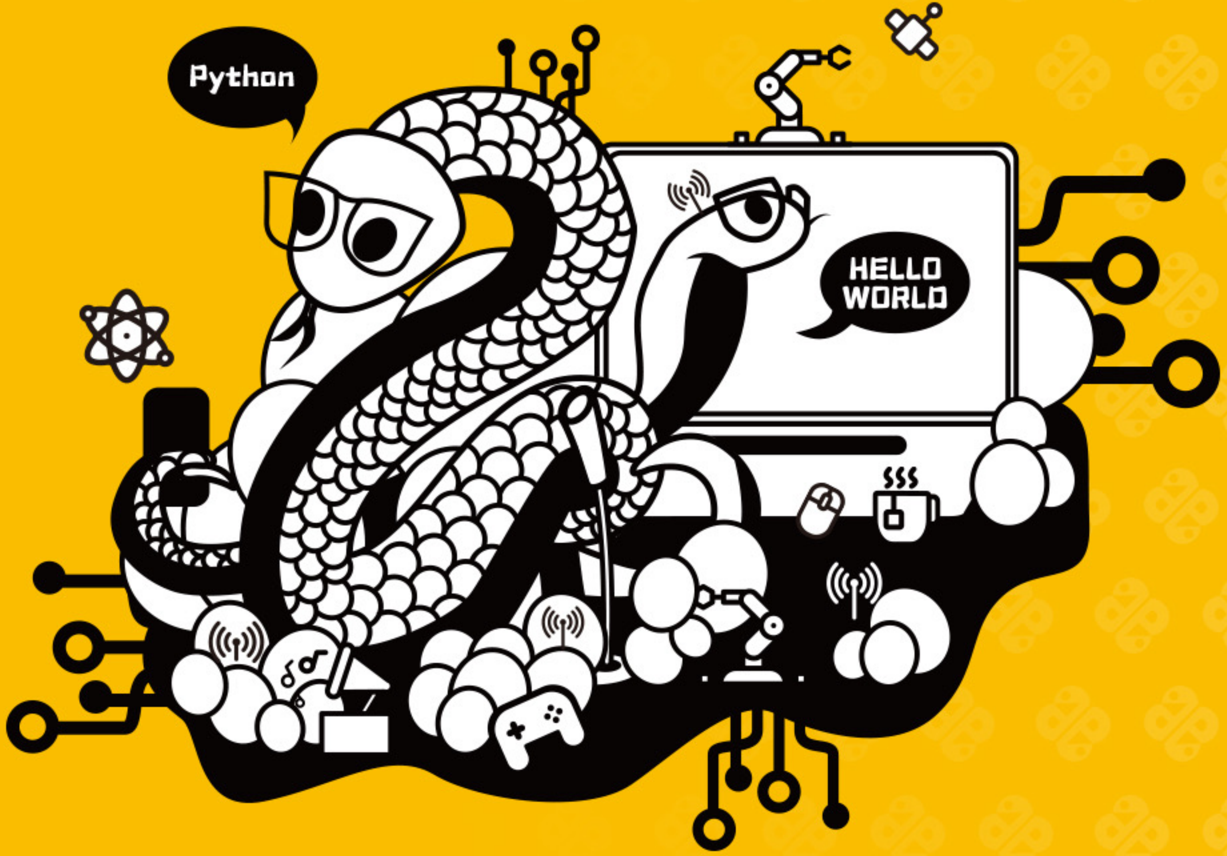
```
root@a559316d3077:/home/andyyang/Documents/project/bert-based-model# ls
README.md    config  entity_relation_extraction.py  networks  pretrain  run.py          run_train.sh  utils
__pycache__  data    evaluation                     output    raw_data  run_predict.sh  tools
root@a559316d3077:/home/andyyang/Documents/project/bert-based-model# 
```

参考文献

[1] Global Pointer：用统一的方式处理嵌套和非嵌套NER

[2] 新冠肺炎（COVID-19）知识图谱

[3] Triton-inference-server



Thanks!

感谢观看