

АНСАМБЛИ В TRITON INFERENCE SERVER

AGENDA

01

Пайплайн инференса



02

Ансамблирование пайплайна



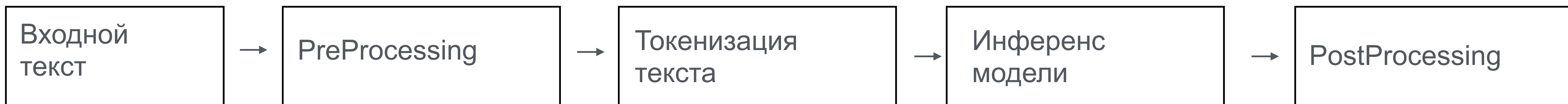
03

Пример ансамбля



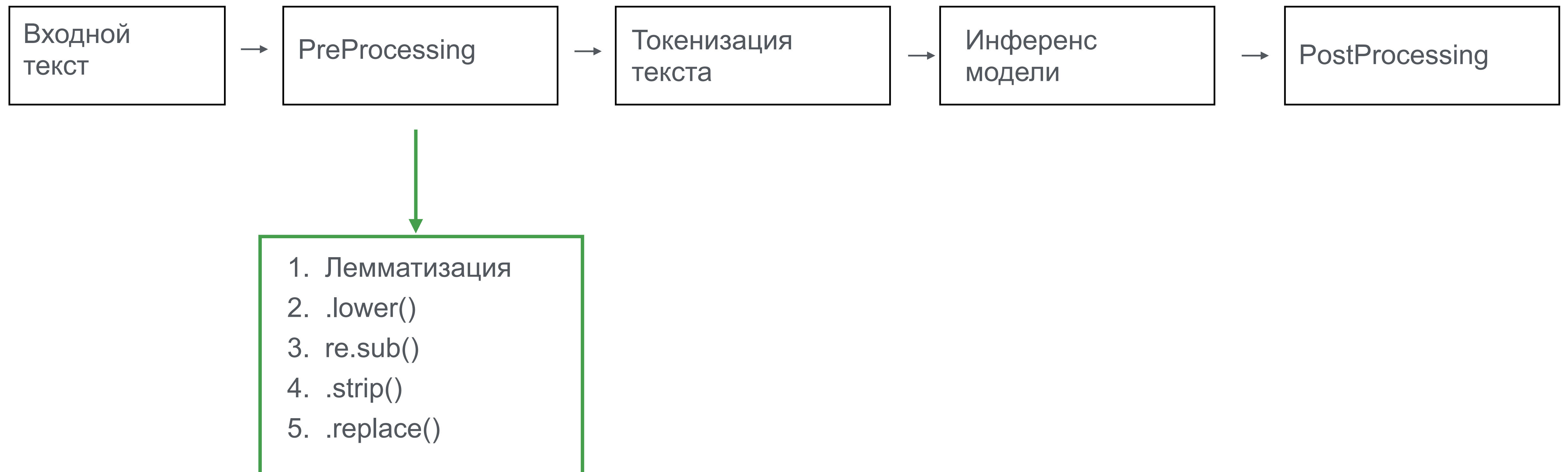
Пайплайн инференса

Задача классификации текста



Пайплайн инференса

Задача классификации текста



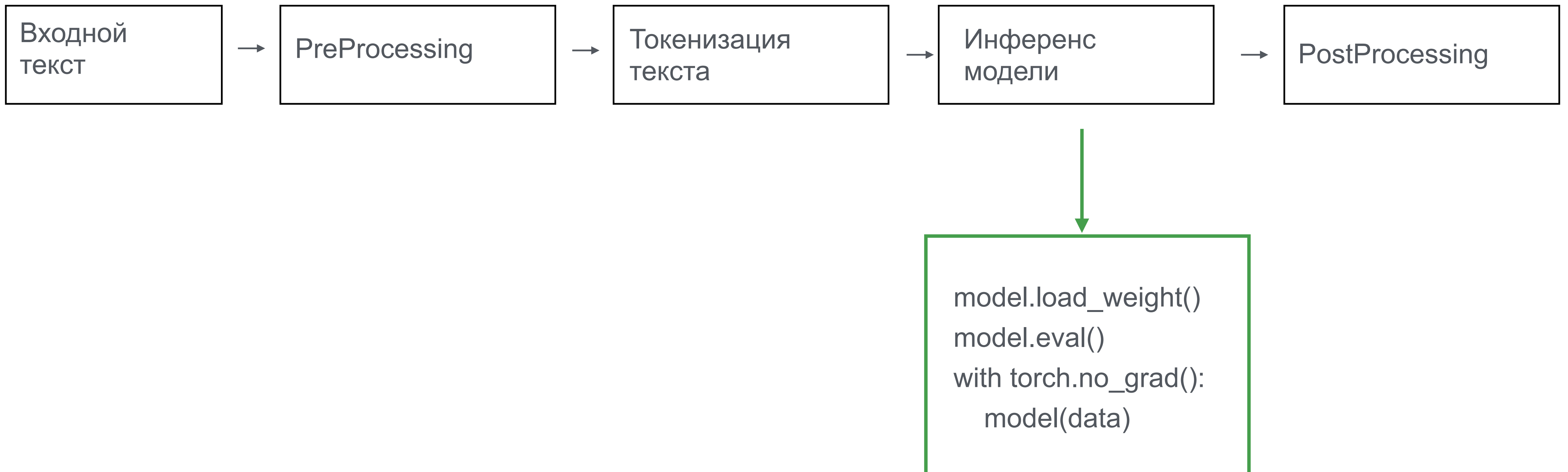
Пайплайн инференса

Задача классификации текста



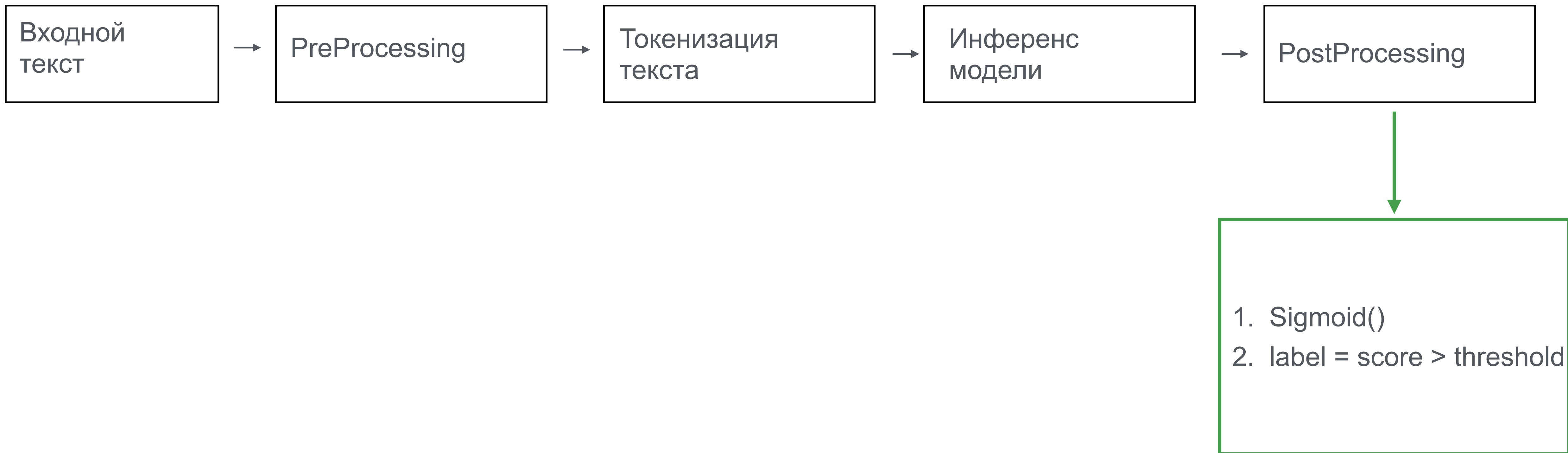
Пайплайн инференса

Задача классификации текста



Пайплайн инференса

Задача классификации текста



Пайплайн инференса

Задача классификации текста

```
def preprocess(text):  
    return text.lower()  
  
def tokenize(text, tokenizer):  
    return tokenizer(  
        text,  
        max_length=512,  
        truncation=True,  
        padding="max_length",  
        return_tensors="pt"  
    )  
  
def infer_model(tokenized_text, model):  
    with torch.no_grad():  
        outputs = model(**tokenized).logits.detach().cpu()  
    return outputs  
  
def postprocess(logits, threshold):  
    return logits.sigmoid() > threshold  
  
def pipeline(text, model, tokenizer, threshold):  
    preprocessed_text = preprocess(text)  
    tokenized_text = tokenize(preprocessed_text, tokenizer)  
    logits = infer_model(tokenized_text, model)  
    labels = postprocess(logits, threshold)  
    return labels
```


Пайплайн инференса

Задача классификации текста

```
def preprocess(text):  
    return text.lower()  
  
def tokenize(text, tokenizer):  
    return tokenizer(  
        text,  
        max_length=512,  
        truncation=True,  
        padding="max_length",  
        return_tensors="pt"  
    )  
  
def infer_model(tokenized_text, model):  
    with torch.no_grad():  
        outputs = model(**tokenized).logits.detach().cpu()  
    return outputs  
  
def postprocess(logits, threshold):  
    return logits.sigmoid() > threshold  
  
def pipeline(text, model, tokenizer, threshold):  
    preprocessed_text = preprocess(text)  
    tokenized_text = tokenize(preprocessed_text, tokenizer)  
    logits = infer_model(tokenized_text, model)  
    labels = postprocess(logits, threshold)  
    return labels
```

1. Как масштабировать?
 1. Токенизатор работает намного быстрее инференса модели

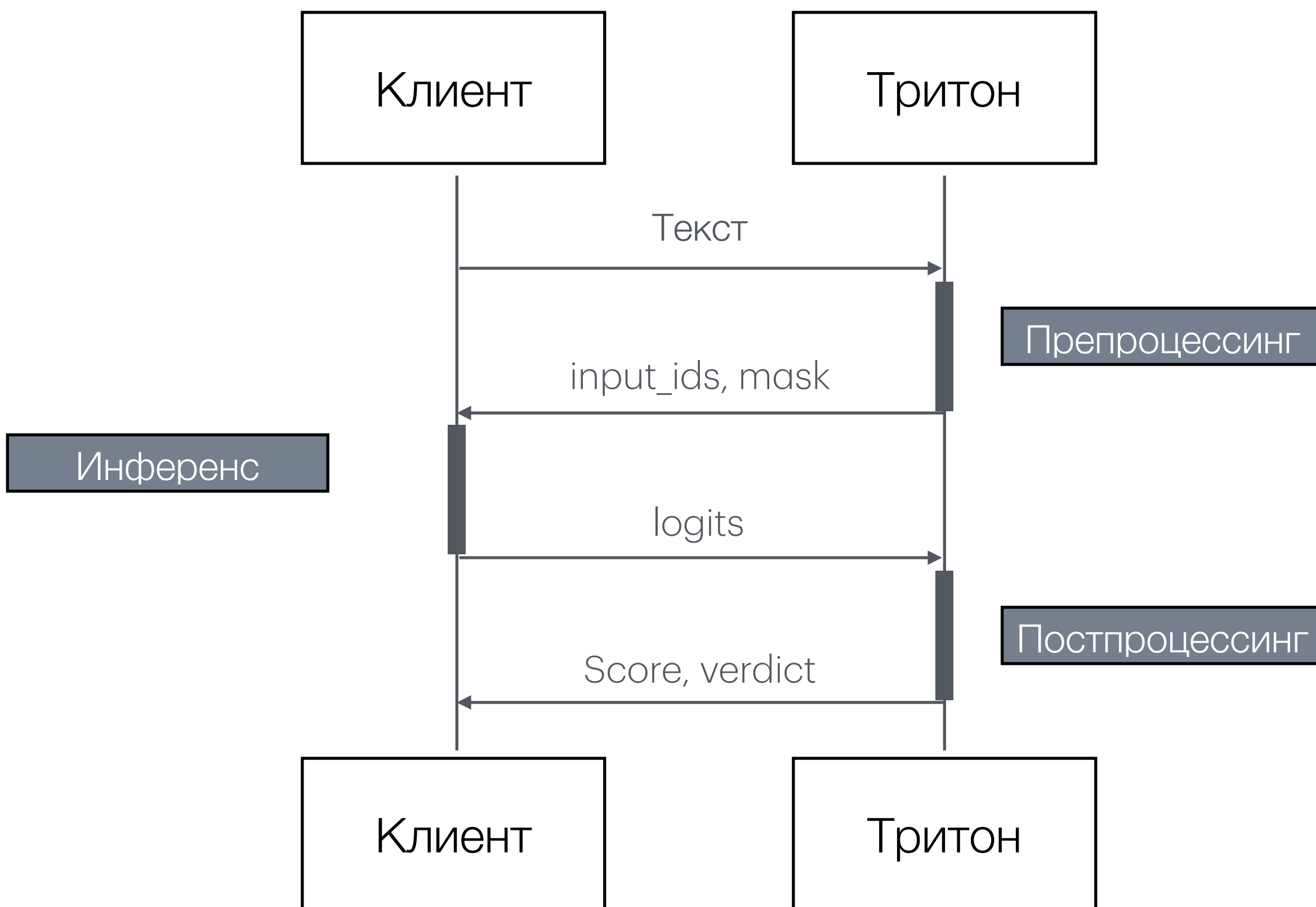
Пайплайн инференса

Задача классификации текста

```
def preprocess(text):  
    return text.lower()  
  
def tokenize(text, tokenizer):  
    return tokenizer(  
        text,  
        max_length=512,  
        truncation=True,  
        padding="max_length",  
        return_tensors="pt"  
    )  
  
def infer_model(tokenized_text, model):  
    with torch.no_grad():  
        outputs = model(**tokenized).logits.detach().cpu()  
    return outputs  
  
def postprocess(logits, threshold):  
    return logits.sigmoid() > threshold  
  
def pipeline(text, model, tokenizer, threshold):  
    preprocessed_text = preprocess(text)  
    tokenized_text = tokenize(preprocessed_text, tokenizer)  
    logits = infer_model(tokenized_text, model)  
    labels = postprocess(logits, threshold)  
    return labels
```

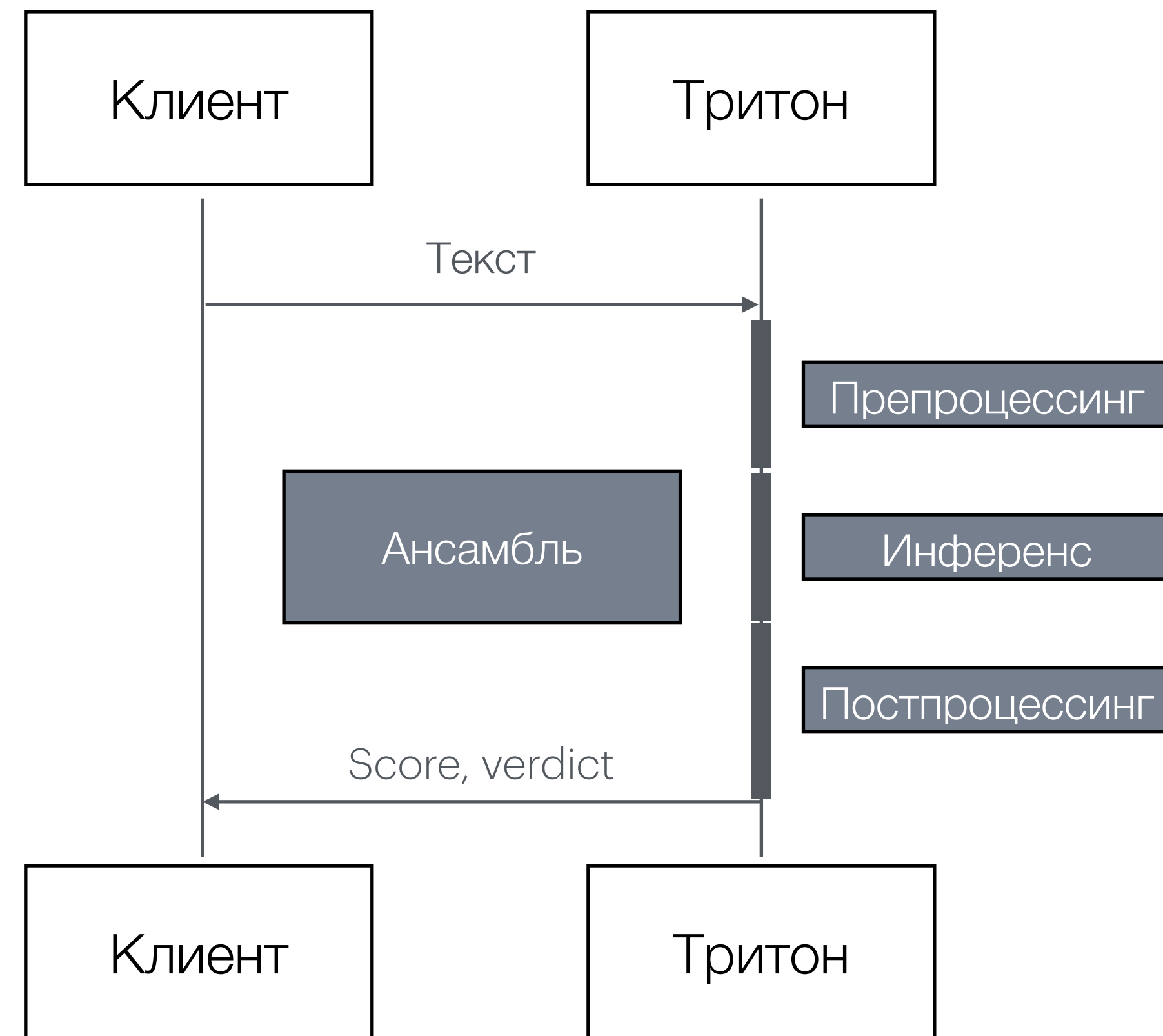
1. Как масштабировать?
 1. Токенизатор работает намного быстрее инференса модели
 2. Неочевидно, как использовать несколько GPU

Ансамблирование пайплайна



1. Итеративно вызываем каждую модель.
2. Нужно сохранять промежуточные результаты в памяти или на диске.

Ансамблирование пайплайна



1. Делаем один вызов ансамбля.
2. Не нужно сохранять промежуточные результаты, сразу получаем желаемое.

Ансамблирование пайплайна

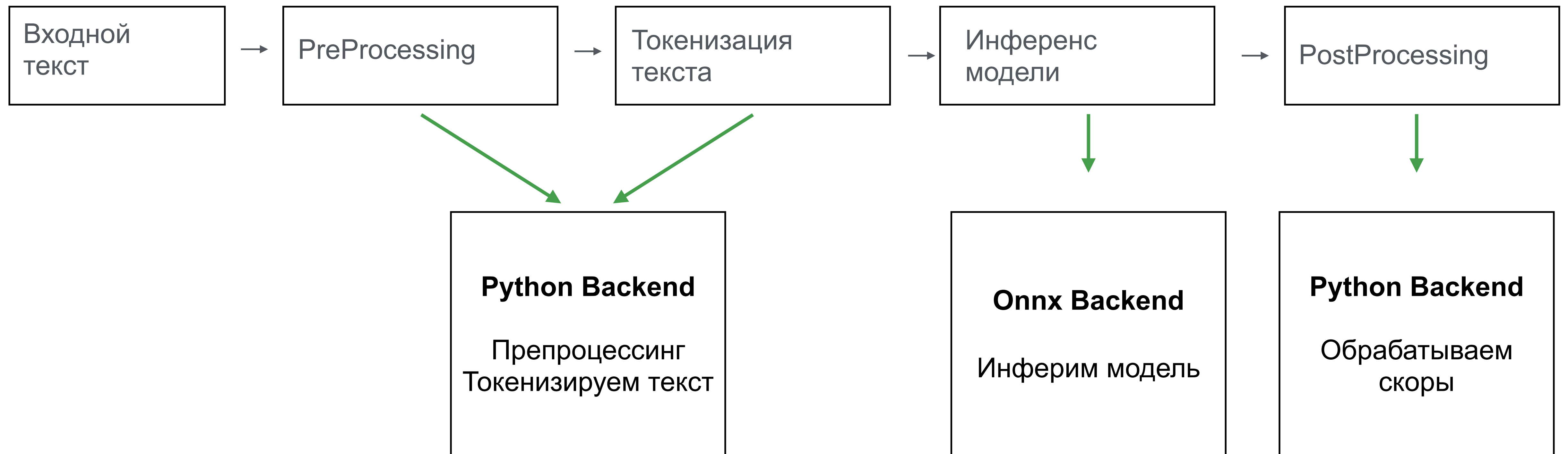
Задача классификации текста

Ансамбль — это система, построенная из одной и более моделей (пайплайн), представленная в виде **направленного графа** без циклов. Представлять решение в виде ансамбля следует, когда не удаётся всю логику собрать на одном бэкенде или появляются узкие места, избавиться от которых можно только с помощью увеличения числа экземпляров отдельных моделей.

Сам по себе ансамбль — это просто спецификация (конфигурационный файл), определяющая порядок передачи данных между моделями с **консистентными** входами и выходами.

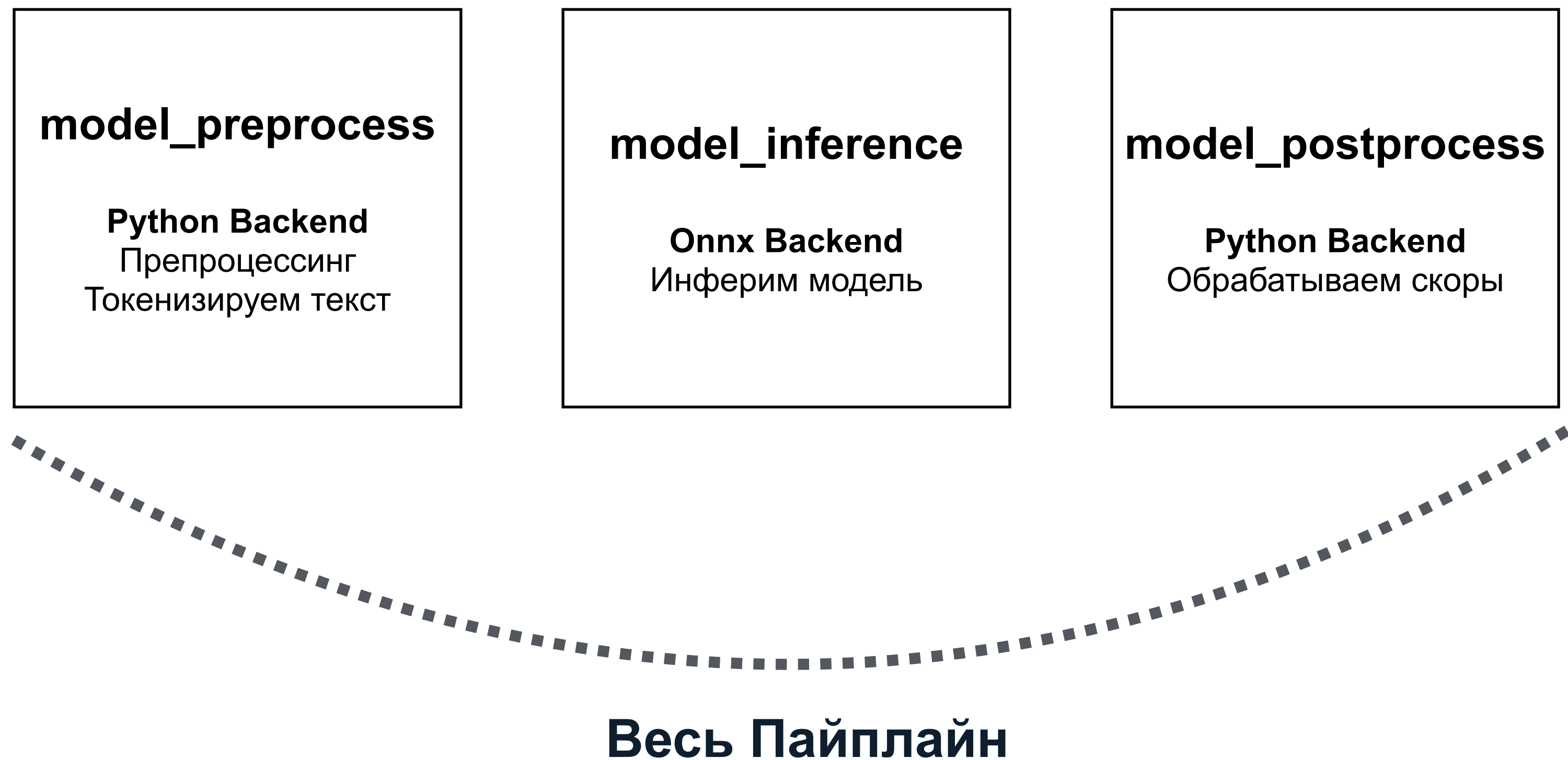
Ансамблирование пайплайна

Задача классификации текста



Ансамблирование пайплайна

Задача классификации текста



Ансамблирование пайплайна

Задача классификации текста

model_preprocess

Python Backend

Препроцессинг
Токенизируем текст

```
name: "model_preprocessing"
backend: "python"
max_batch_size: 0

input [
  {
    name: "TEXT"
    data_type: TYPE_STRING
    dims: [ -1, 1 ]
  }
]

output [
  {
    name: "input_ids"
    data_type: TYPE_INT64
    dims: [ -1, 512 ]
  },
  {
    name: "attention_mask"
    data_type: TYPE_INT64
    dims: [ -1, 512 ]
  }
]

instance_group [
  {
    count: 4
    kind: KIND_CPU
  }
]

parameters: {
  key: "EXECUTION_ENV_PATH",
  value: {string_value: "$$TRITON_MODEL_DIRECTORY/triton_env.tar.gz"}
}
```


Ансамблирование пайплайна

Задача классификации текста

model_inference

Onnx Backend
Инферим модель

```
name: "model_inference"
platform: "onnxruntime_onnx"
max_batch_size: 0

input [
  {
    name: "input_ids"
    data_type: TYPE_INT64
    dims: [ -1, 512 ]
  },
  {
    name: "attention_mask"
    data_type: TYPE_INT64
    dims: [ -1, 512 ]
  }
]

output [
  {
    name: "logits"
    data_type: TYPE_FP32
    dims: [ -1, 1 ]
  }
]

instance_group [
  {
    count: 1
    kind: KIND_GPU
  }
]
```

Ансамблирование пайплайна

Задача классификации текста

model_postprocess

Python Backend
Обрабатываем скоры

```
name: "model_postprocessing"
backend: "python"
max_batch_size: 0

input [
  {
    name: "logits"
    data_type: TYPE_FP32
    dims: [ -1, 1 ]
  },
  {
    name: "THRESHOLD"
    data_type: TYPE_FP32
    dims: [ -1 ]
  }
]

output [
  {
    name: "PROBABILITIES"
    data_type: TYPE_FP32
    dims: [ -1, -1 ]
  },
  {
    name: "VERDICTS"
    data_type: TYPE_BOOL
    dims: [ -1, -1 ]
  }
]

instance_group [
  {
    count: 4
    kind: KIND_CPU
  }
]
```

Ансамблирование пайплайна

Задача классификации текста



model_ensemble

```
name: "model_ensemble"
max_batch_size: 0
platform: "ensemble"

input [
  {
    name: "TEXT"
    data_type: TYPE_STRING
    dims: [ -1, 1 ]
  },
  {
    name: "TRUE_THRESHOLD"
    data_type: TYPE_FP32
    dims: [ -1 ]
  }
]

output [
  {
    name: "PROBABILITIES"
    data_type: TYPE_FP32
    dims: [ -1, -1 ]
  },
  {
    name: "VERDICTS"
    data_type: TYPE_BOOL
    dims: [ -1, -1 ]
  }
]
```

Ансамблирование пайплайна

Задача классификации текста

Ансамбль - спецификация, определяющая порядок передачи данных между моделями **с КОНСИСТЕНТНЫМИ** входами и выходами.



model_ensemble

```
name: "model_ensemble"
max_batch_size: 0
platform: "ensemble"

input [
  {
    name: "TEXT"
    data_type: TYPE_STRING
    dims: [ -1, 1 ]
  },
  {
    name: "TRUE_THRESHOLD"
    data_type: TYPE_FP32
    dims: [ -1 ]
  }
]

output [
  {
    name: "PROBABILITIES"
    data_type: TYPE_FP32
    dims: [ -1, -1 ]
  },
  {
    name: "VERDICTS"
    data_type: TYPE_BOOL
    dims: [ -1, -1 ]
  }
]
```

Ансамблирование пайплайна

Задача классификации текста

Ансамбль - спецификация,
определяющая порядок передачи
данных между моделями
с **консистентными входами**.

```
ensemble_scheduling {  
  step [  
    {  
      model_name: "model_name"  
      model_version: 1  
      input_map [  
        {  
          key: "INPUT_1"  
          value: "INPUT_1"  
        }  
      ]  
      output_map [  
        {  
          key: "OUTPUT_1"  
          value: "OUTPUT_1"  
        },  
        {  
          key: "OUTPUT_2"  
          value: "OUTPUT_2"  
        }  
      ]  
    },  
    {  
      model_name: "another_model"  
      ...  
    },  
  ]  
}
```

Ансамблирование пайплайна

Задача классификации текста



Ансамбль - спецификация, определяющая порядок передачи данных между моделями **с консистентными входами**.

```
ensemble_scheduling {  
  step [  
    {  
      model_name: "model_name" ← Название модели  
      model_version: 1  
      input_map [  
        {  
          key: "INPUT_1"  
          value: "INPUT_1"  
        }  
      ]  
      output_map [  
        {  
          key: "OUTPUT_1"  
          value: "OUTPUT_1"  
        },  
        {  
          key: "OUTPUT_2"  
          value: "OUTPUT_2"  
        }  
      ]  
    },  
    {  
      model_name: "another_model"  
      ...  
    },  
  ]  
}
```

Ансамблирование пайплайна

Задача классификации текста




Ансамбль - спецификация, определяющая порядок передачи данных между моделями **с консистентными входами**.

```
ensemble_scheduling {  
  step [  
    {  
      model_name: "model_name"  Название модели  
      model_version: 1  Версия модели  
      input_map [  
        {  
          key: "INPUT_1"  
          value: "INPUT_1"  
        }  
      ]  
      output_map [  
        {  
          key: "OUTPUT_1"  
          value: "OUTPUT_1"  
        },  
        {  
          key: "OUTPUT_2"  
          value: "OUTPUT_2"  
        }  
      ]  
    },  
    {  
      model_name: "another_model"  
      ...  
    },  
  ]  
}
```

Ансамблирование пайплайна

Задача классификации текста

Ансамбль - спецификация, определяющая порядок передачи данных между моделями **с консистентными входами**.

```
ensemble_scheduling {  
  step [  
    {  
      model_name: "model_name"  Название модели  
      model_version: 1  Версия модели  
      input_map [  
        {  
          key: "INPUT_1"  Сопоставление входных данных  
          value: "INPUT_1"  
        }  
      ]  
      output_map [  
        {  
          key: "OUTPUT_1"  
          value: "OUTPUT_1"  
        },  
        {  
          key: "OUTPUT_2"  
          value: "OUTPUT_2"  
        }  
      ]  
    },  
    {  
      model_name: "another_model"  
      ...  
    },  
  ]  
}
```


Ансамблирование пайплайна

Задача классификации текста

Ансамбль - спецификация, определяющая порядок передачи данных между моделями **с консистентными входами.**

```
ensemble_scheduling {  
  step [  
    {  
      model_name: "model_name"  ← Название модели  
      model_version: 1          ← Версия модели  
      input_map [  
        {  
          key: "INPUT_1"        ← Сопоставление входных данных  
          value: "INPUT_1"  
        }  
      ]  
      output_map [  
        {  
          key: "OUTPUT_1"       ← Сопоставление выходных данных  
          value: "OUTPUT_1"  
        },  
        {  
          key: "OUTPUT_2"  
          value: "OUTPUT_2"  
        }  
      ]  
    },  
    {  
      model_name: "another_model"  
      ...  
    },  
  ]  
}
```

Ансамблирование пайплайна

Задача классификации текста

Ансамбль - спецификация, определяющая порядок передачи данных между моделями **с консистентными входами.**

```
ensemble_scheduling {  
  step [  
    {  
      model_name: "model_name" ← Название модели  
      model_version: 1 ← Версия модели  
      input_map [  
        {  
          key: "INPUT_1" ← Сопоставление входных данных  
          value: "INPUT_1"  
        }  
      ]  
      output_map [  
        {  
          key: "OUTPUT_1" ← Сопоставление выходных данных  
          value: "OUTPUT_1"  
        },  
        {  
          key: "OUTPUT_2"  
          value: "OUTPUT_2"  
        }  
      ]  
    },  
    {  
      model_name: "another_model" ← Следующая компонента ансамбля  
      ...  
    }  
  ]  
}
```

Ансамблирование пайплайна

Задача классификации текста

```
name: "model_ensemble"
max_batch_size: 0
platform: "ensemble"

input [
  {
    name: "TEXT"
    data_type: TYPE_STRING
    dims: [ -1, 1 ]
  },
  {
    name: "TRUE_THRESHOLD"
    data_type: TYPE_FP32
    dims: [ -1 ]
  }
]

output [
  {
    name: "PROBABILITIES"
    data_type: TYPE_FP32
    dims: [ -1, -1 ]
  },
  {
    name: "VERDICTS"
    data_type: TYPE_BOOL
    dims: [ -1, -1 ]
  }
]
```

model_preprocess

Python Backend
Препроцессинг
Токенизируем текст

```
{
  model_name: "model_preprocessing"
  model_version: 1
  input_map [
    {
      key: "TEXT"
      value: "TEXT"
    }
  ]
  output_map [
    {
      key: "input_ids"
      value: "input_ids"
    },
    {
      key: "attention_mask"
      value: "attention_mask"
    }
  ]
}
```

model_inference

Onnx Backend
Инферим модель

model_postprocess

Python Backend
Обрабатываем скоры

Ансамблирование пайплайна

Задача классификации текста

```
name: "model_ensemble"
max_batch_size: 0
platform: "ensemble"

input [
  {
    name: "TEXT"
    data_type: TYPE_STRING
    dims: [ -1, 1 ]
  },
  {
    name: "TRUE_THRESHOLD"
    data_type: TYPE_FP32
    dims: [ -1 ]
  }
]

output [
  {
    name: "PROBABILITIES"
    data_type: TYPE_FP32
    dims: [ -1, -1 ]
  },
  {
    name: "VERDICTS"
    data_type: TYPE_BOOL
    dims: [ -1, -1 ]
  }
]
```

model_preprocess

Python Backend
Препроцессинг
Токенизируем текст

```
{
  model_name: "model_preprocessing"
  model_version: 1
  input_map [
    {
      key: "TEXT"
      value: "TEXT"
    }
  ]
  output_map [
    {
      key: "input_ids"
      value: "input_ids"
    },
    {
      key: "attention_mask"
      value: "attention_mask"
    }
  ]
}
```

model_inference

Onnx Backend
Инферим модель

```
{
  model_name: "model_inference"
  model_version: 1
  input_map [
    {
      key: "input_ids"
      value: "input_ids"
    },
    {
      key: "attention_mask"
      value: "attention_mask"
    }
  ]
  output_map [
    {
      key: "logits"
      value: "logits"
    }
  ]
}
```

model_postprocess

Python Backend
Обрабатываем скоры

Ансамблирование пайплайна

Задача классификации текста

```
name: "model_ensemble"
max_batch_size: 0
platform: "ensemble"

input [
  {
    name: "TEXT"
    data_type: TYPE_STRING
    dims: [ -1, 1 ]
  },
  {
    name: "TRUE_THRESHOLD"
    data_type: TYPE_FP32
    dims: [ -1 ]
  }
]

output [
  {
    name: "PROBABILITIES"
    data_type: TYPE_FP32
    dims: [ -1, -1 ]
  },
  {
    name: "VERDICTS"
    data_type: TYPE_BOOL
    dims: [ -1, -1 ]
  }
]
```

model_preprocess

Python Backend
Препроцессинг
Токенизируем текст

```
{
  model_name: "model_preprocessing"
  model_version: 1
  input_map [
    {
      key: "TEXT"
      value: "TEXT"
    }
  ]
  output_map [
    {
      key: "input_ids"
      value: "input_ids"
    },
    {
      key: "attention_mask"
      value: "attention_mask"
    }
  ]
}
```

model_inference

Onnx Backend
Инферим модель

```
{
  model_name: "model_inference"
  model_version: 1
  input_map [
    {
      key: "input_ids"
      value: "input_ids"
    },
    {
      key: "attention_mask"
      value: "attention_mask"
    }
  ]
  output_map [
    {
      key: "logits"
      value: "logits"
    }
  ]
}
```

model_postprocess

Python Backend
Обрабатываем скоры

```
{
  model_name: "model_postprocessing"
  model_version: 1
  input_map [
    {
      key: "TRUE_THRESHOLD"
      value: "TRUE_THRESHOLD"
    },
    {
      key: "logits"
      value: "logits"
    }
  ]
  output_map [
    {
      key: "VERDICTS"
      value: "VERDICTS"
    },
    {
      key: "PROBABILITIES"
      value: "PROBABILITIES"
    }
  ]
}
```

Ансамблирование пайплайна

Задача классификации текста

```
name: "model_ensemble"
max_batch_size: 0
platform: "ensemble"

input [
  {
    name: "TEXT"
    data_type: TYPE_STRING
    dims: [ -1, 1 ]
  },
  {
    name: "TRUE_THRESHOLD"
    data_type: TYPE_FP32
    dims: [ -1 ]
  }
]

output [
  {
    name: "PROBABILITIES"
    data_type: TYPE_FP32
    dims: [ -1, -1 ]
  },
  {
    name: "VERDICTS"
    data_type: TYPE_BOOL
    dims: [ -1, -1 ]
  }
]
```

model_postprocess

Python Backend
Обрабатываем скоры

```
{
  model_name: "model_postprocessing"
  model_version: 1
  input_map [
    {
      key: "TRUE_THRESHOLD"
      value: "TRUE_THRESHOLD"
    },
    {
      key: "logits"
      value: "logits"
    }
  ]
  output_map [
    {
      key: "VERDICTS"
      value: "VERDICTS"
    },
    {
      key: "PROBABILITIES"
      value: "PROBABILITIES"
    }
  ]
}
```

В step'е ансамбля мы нигде не указываем тип данных и размерность, только в конфиге.
Размерность и тип выходов step'а берутся из конфига каждого шага отдельно.

Ансамблирование пайплайна

Задача классификации текста

В step'е ансамбля мы нигде не указываем тип данных и размерность, только в конфиге.

Размерность и тип выходов step'а берутся из конфига каждого шага отдельно.

```
{
  model_name: "model_postprocessing"
  model_version: 1
  input_map [
    {
      key: "TRUE_THRESHOLD"
      value: "TRUE_THRESHOLD"
    },
    {
      key: "logits"
      value: "logits"
    }
  ]
  output_map [
    {
      key: "VERDICTS"
      value: "VERDICTS"
    },
    {
      key: "PROBABILITIES"
      value: "PROBABILITIES"
    }
  ]
}
```

```
name: "model_postprocessing"
backend: "python"
max_batch_size: 0

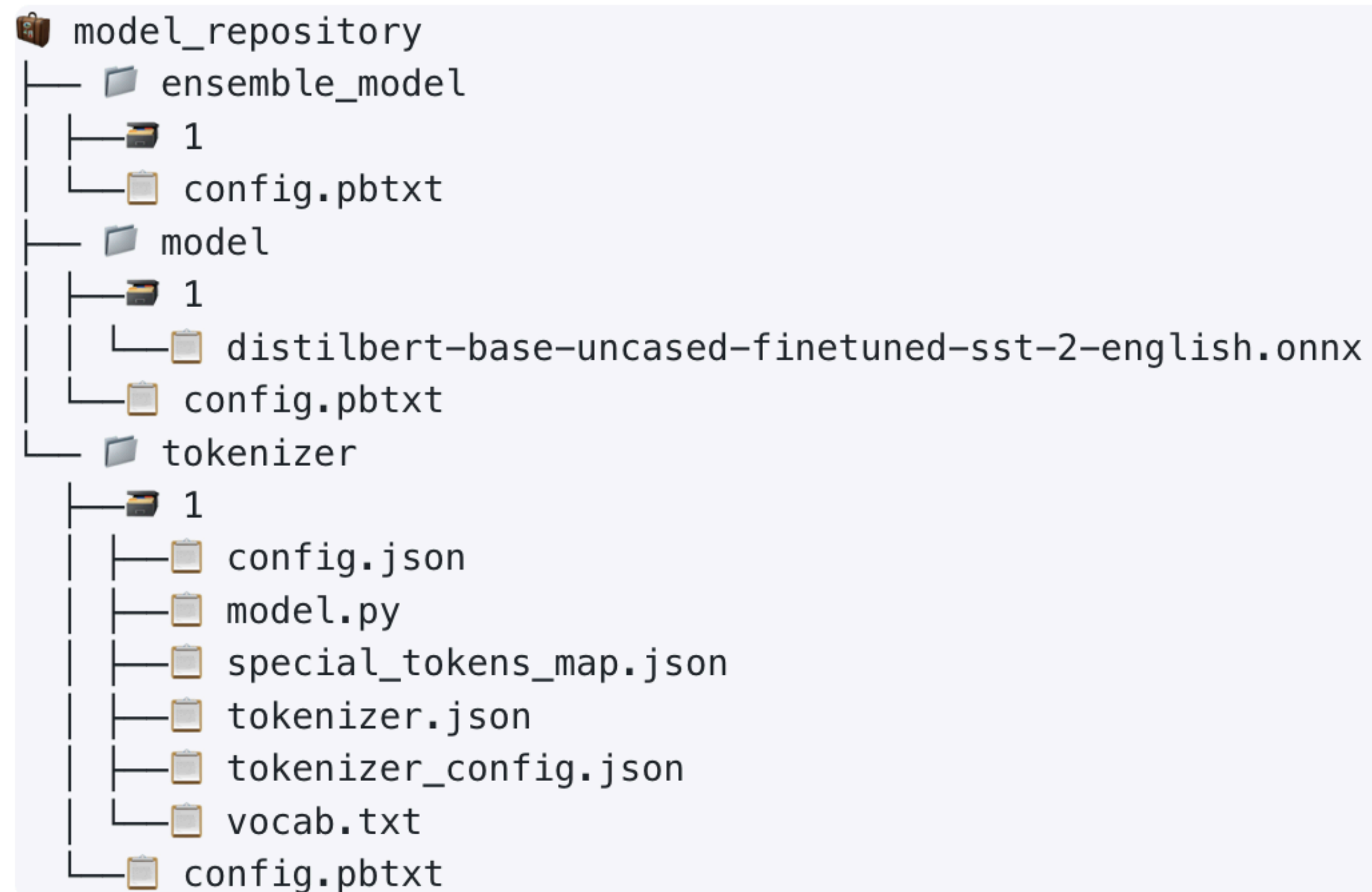
input [
  {
    name: "logits"
    data_type: TYPE_FP32
    dims: [ -1, 1 ]
  },
  {
    name: "THRESHOLD"
    data_type: TYPE_FP32
    dims: [ -1 ]
  }
]

output [
  {
    name: "PROBABILITIES"
    data_type: TYPE_FP32
    dims: [ -1, -1 ]
  },
  {
    name: "VERDICTS"
    data_type: TYPE_BOOL
    dims: [ -1, -1 ]
  }
]

instance_group [
  {
    count: 4
    kind: KIND_CPU
  }
]
```

Ансамблирование пайплайна

Задача классификации текста



Ансамблирование пайплайна

Ансамбль может включать в себя несколько моделей

Photo OCR pipeline

1. Text detection



1.

detection_preprocess

Python Backend

detection_inference

Onnx Backend

Ансамблирование пайплайна

Ансамбль может включать в себя несколько моделей

Photo OCR pipeline

1. Text detection



2. Character segmentation



1.

detection_preprocess

Python Backend

detection_inference

Onnx Backend

2.

detection_postprocess

Python Backend

- Crop Bounding Box
- Align Bounding Box

recognition_preprocess

Python Backend
Image resize + normalize

Ансамблирование пайплайна

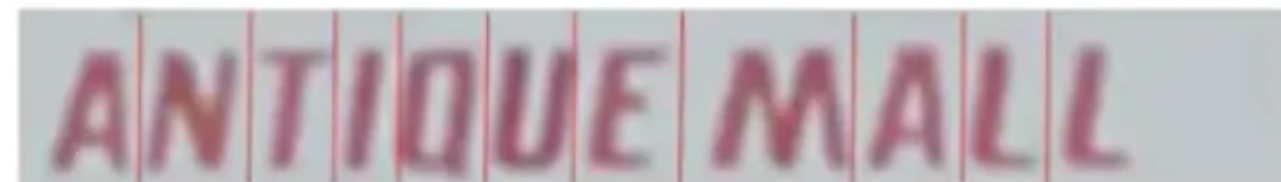
Ансамбль может включать в себя несколько моделей

Photo OCR pipeline

1. Text detection



2. Character segmentation



3. Character classification



1.

detection_preprocess

Python Backend

detection_inference

Onnx Backend

2.

detection_postprocess

Python Backend
- Crop Bounding Box
- Align Bounding Box

recognition_preprocess

Python Backend
Image resize + normalize

3.

recognition_inference

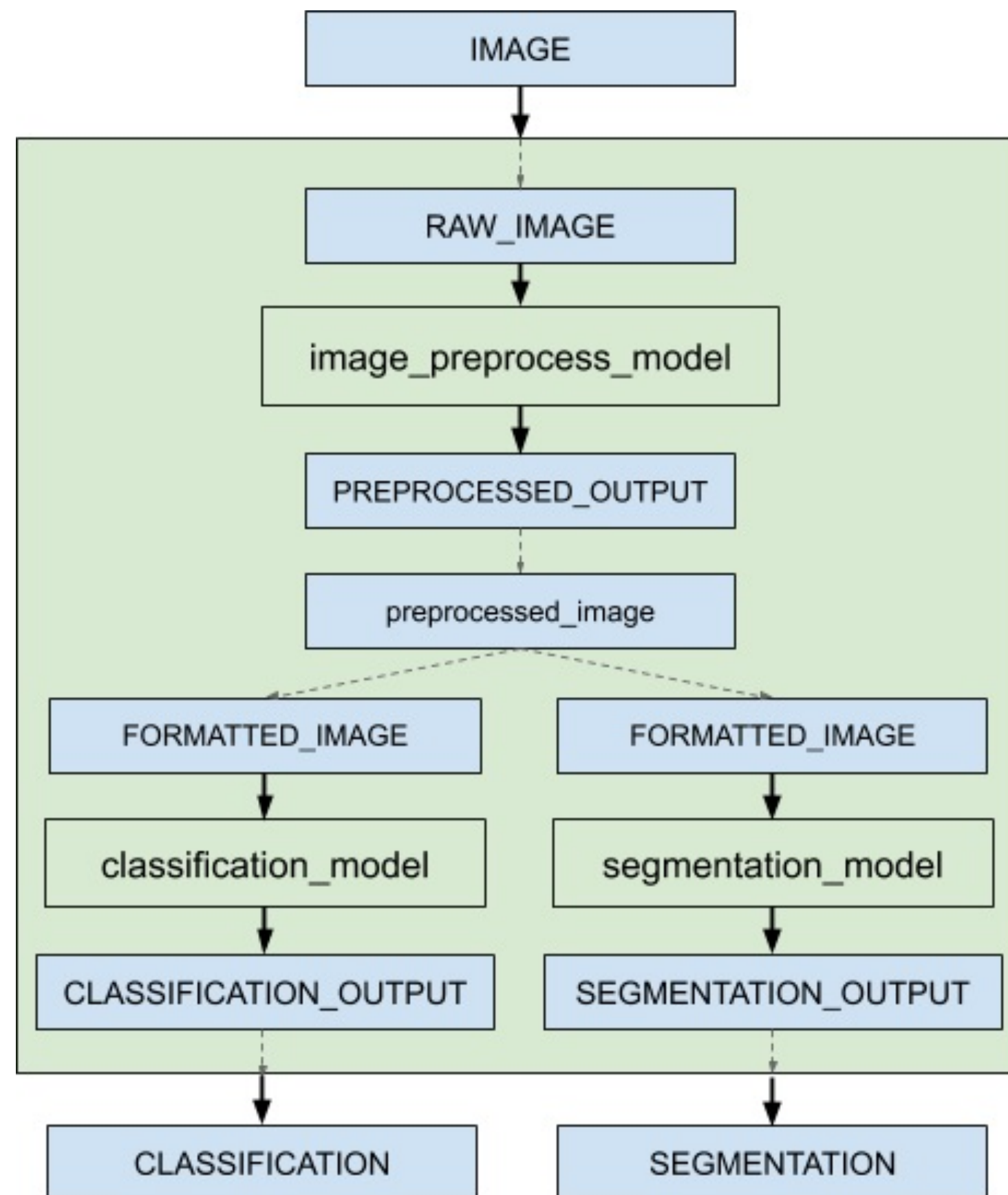
Onnx Backend

recognition_postprocess

Onnx Backend

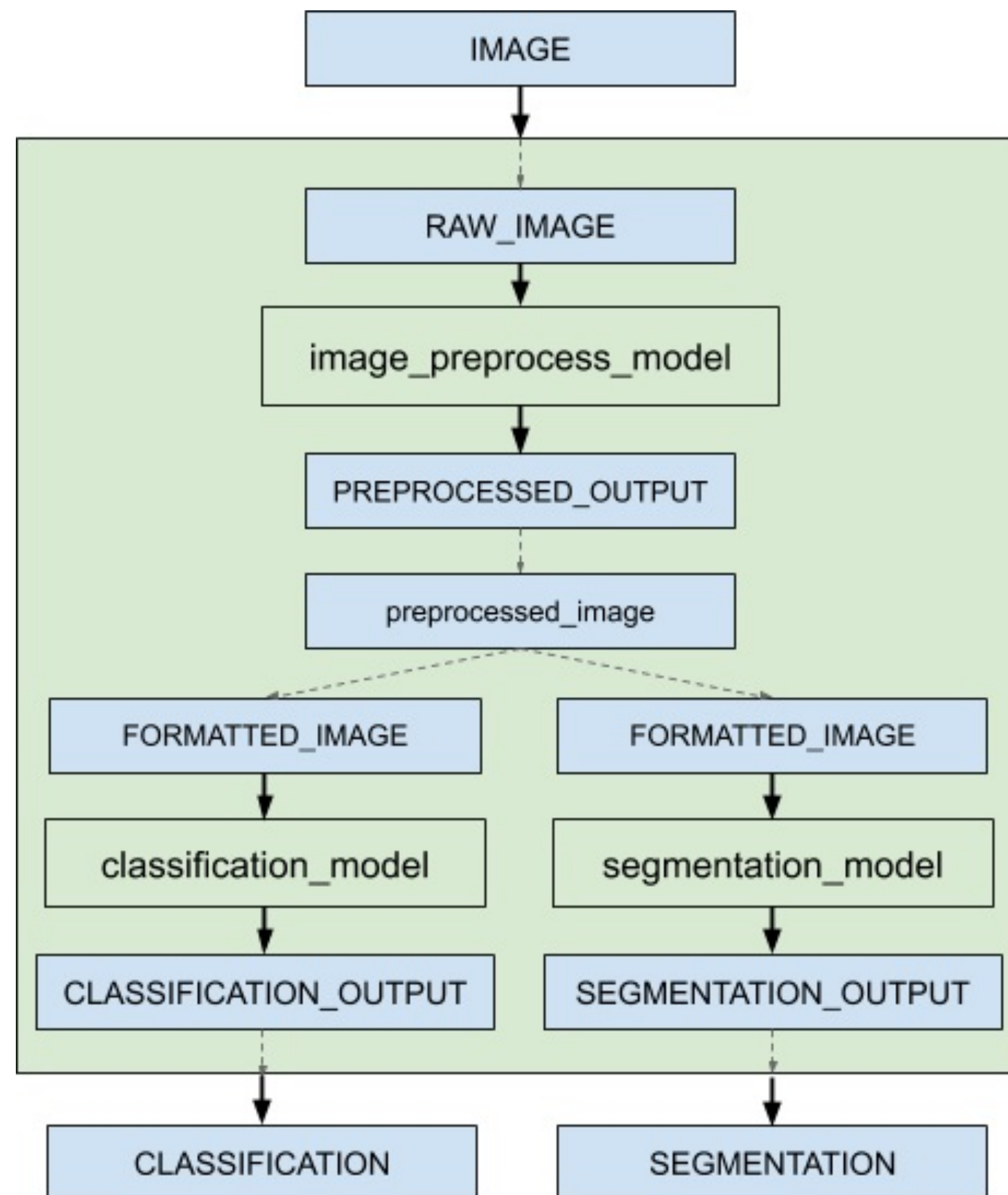
Ансамблирование пайплайна

Ансамбль может иметь параллельную структуру.



Ансамблирование пайплайна

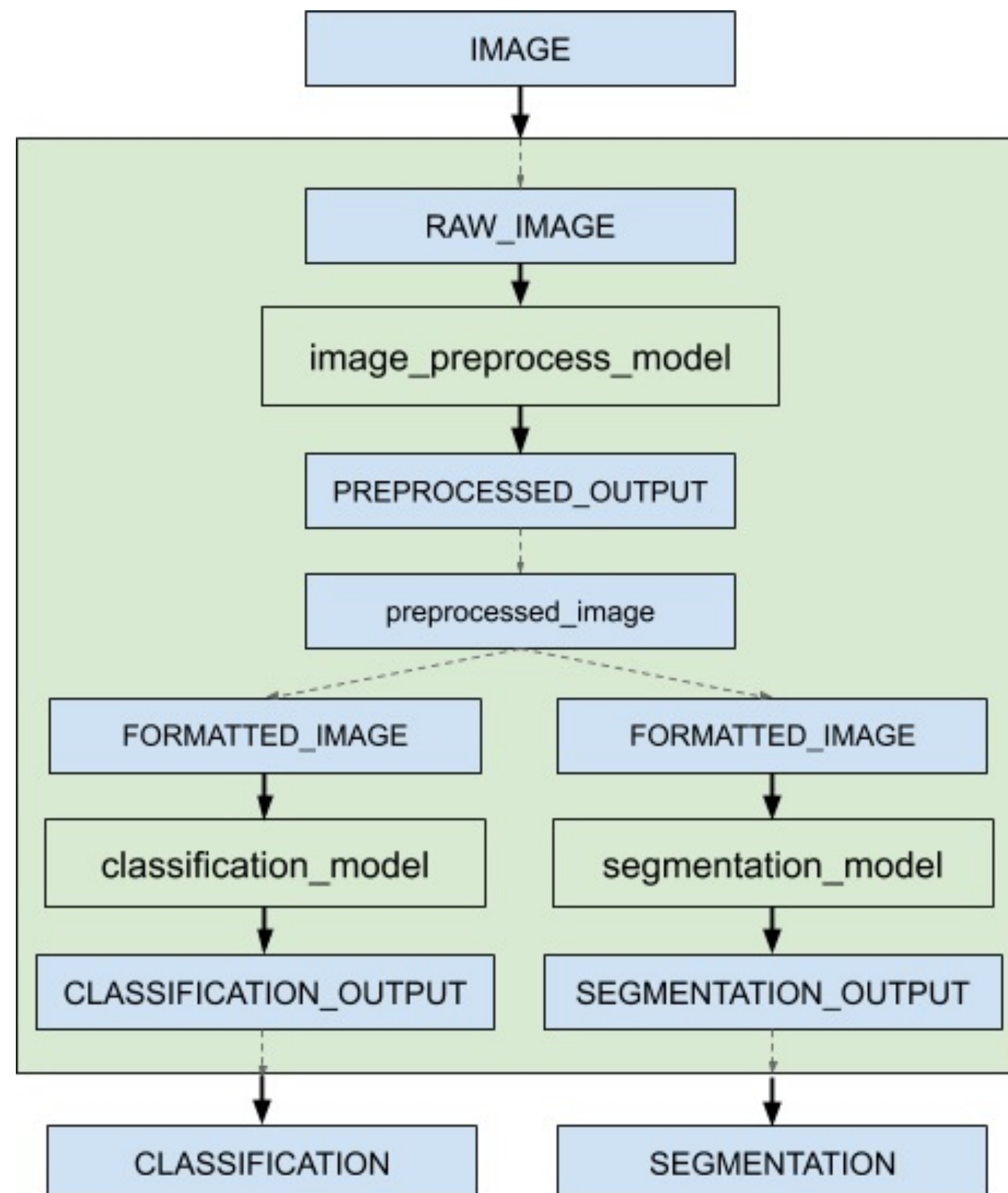
Ансамбль может иметь параллельную структуру.



1. На вход в ансамбль приходит изображение («IMAGE» tensor), которое сопоставляется input «RAW_IMAGE».

Ансамблирование пайплайна

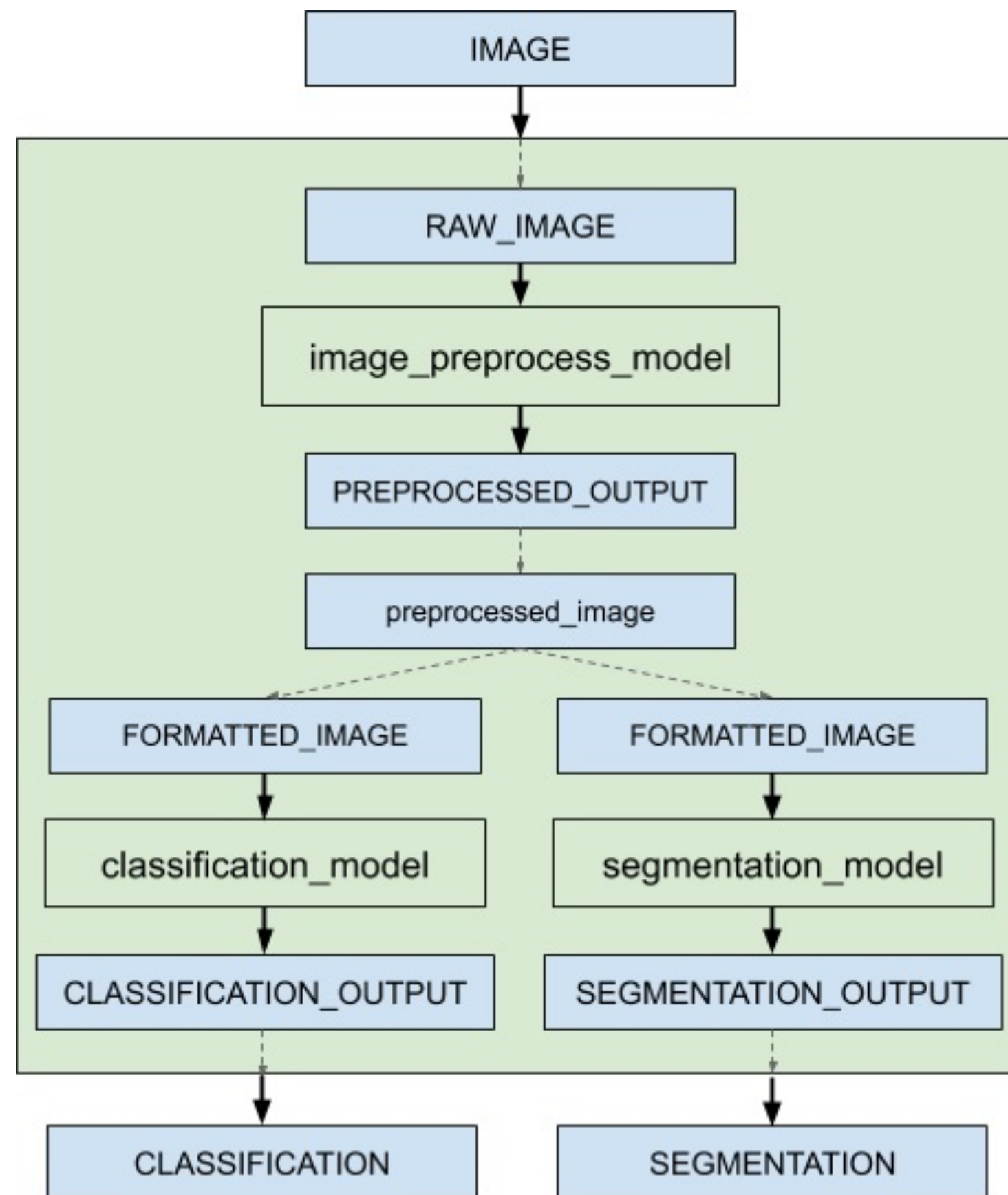
Ансамбль может иметь параллельную структуру.



1. На вход в ансамбль приходит изображение («IMAGE» tensor), которое сопоставляется input «RAW_IMAGE».
2. Отправляется внутренний запрос в модель preprocessing.

Ансамблирование пайплайна

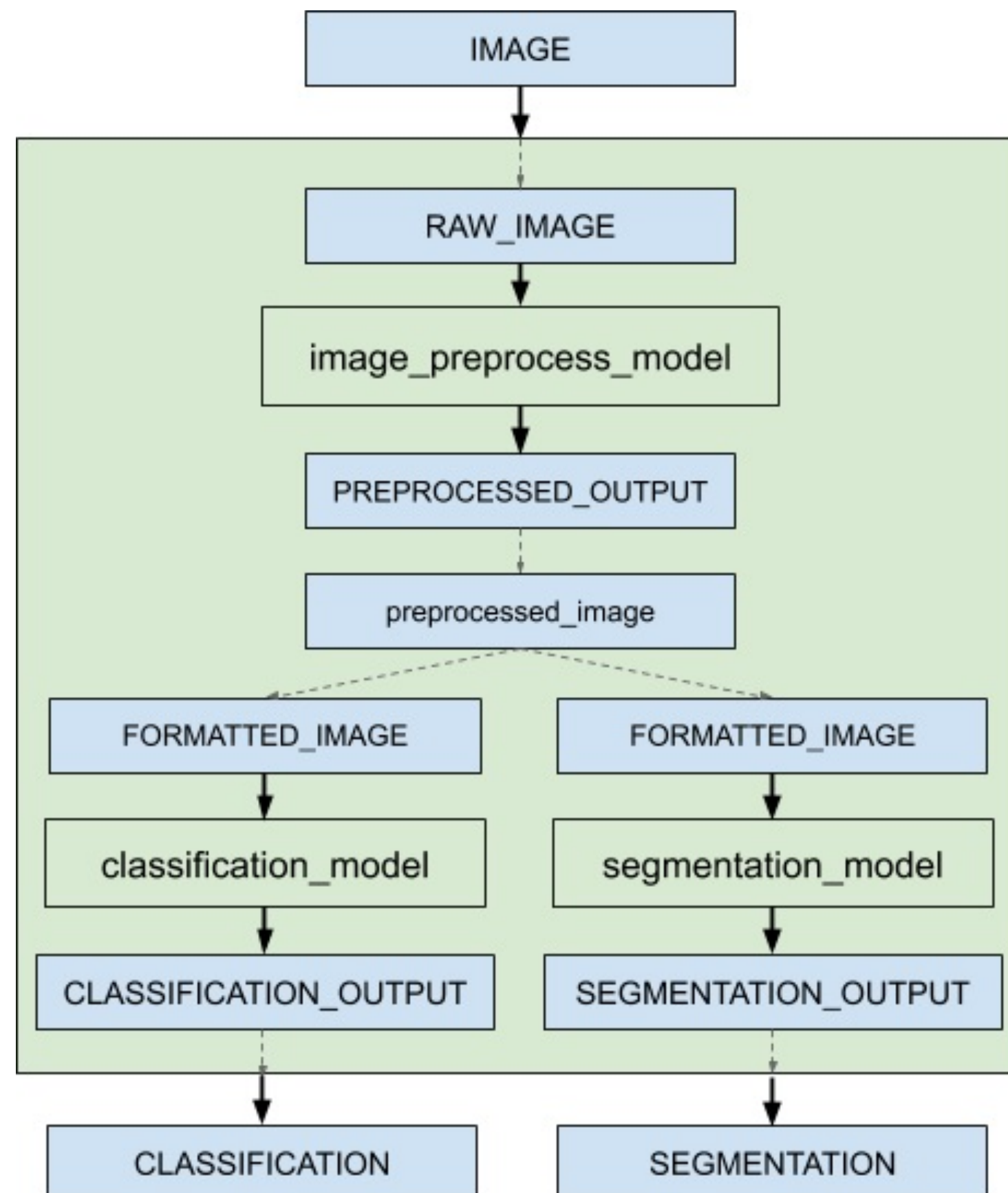
Ансамбль может иметь параллельную структуру.



1. На вход в ансамбль приходит изображение («IMAGE» tensor), которое сопоставляется input «RAW_IMAGE».
2. Отправляется внутренний запрос в модель preprocessing.
3. Получаем предобработанное изображение «PREPROCESSED_OUTPUT» и мапим в «preprocess_image».

Ансамблирование пайплайна

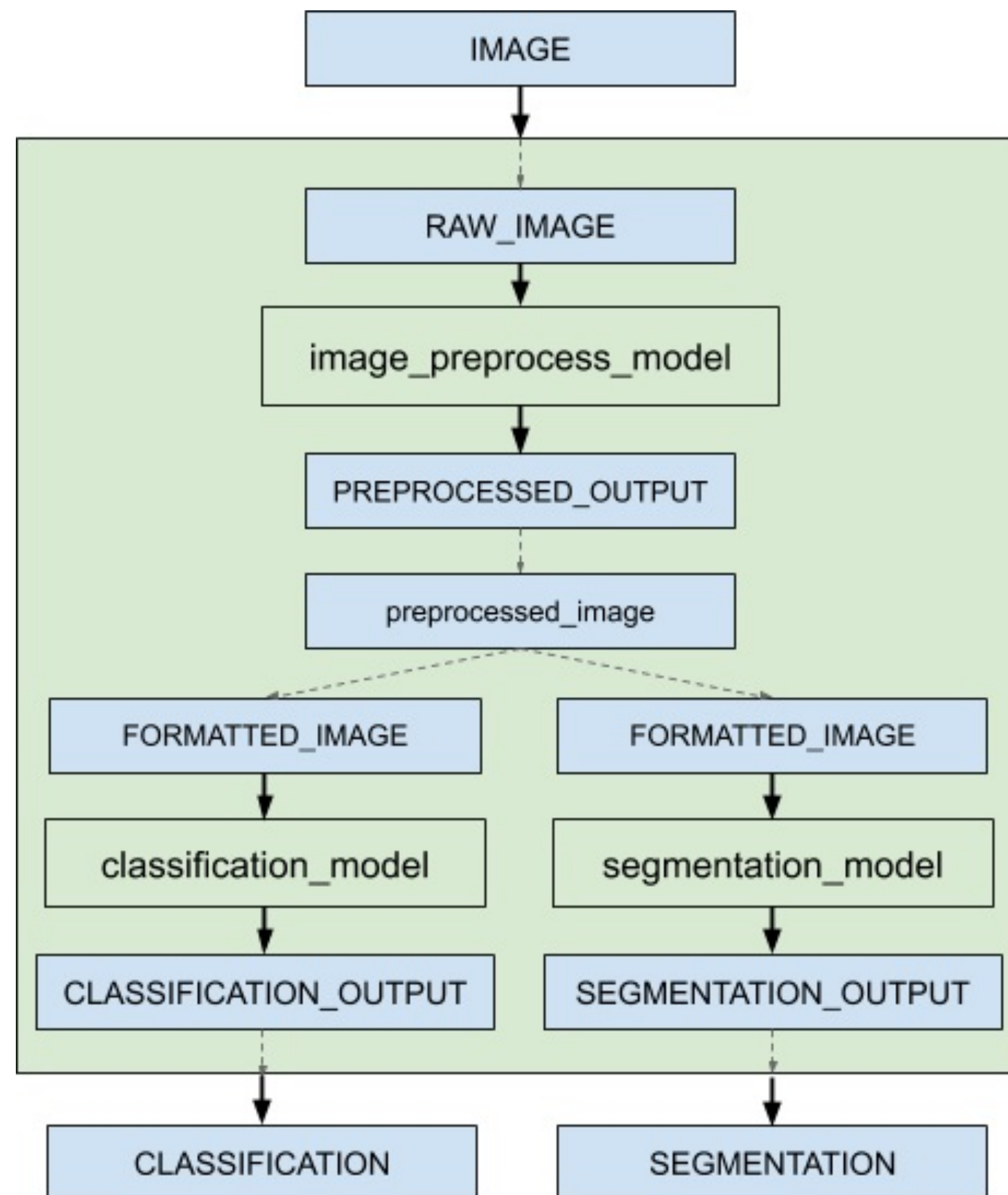
Ансамбль может иметь параллельную структуру.



1. На вход в ансамбль приходит изображение («IMAGE» tensor), которое сопоставляется input «RAW_IMAGE».
2. Отправляется внутренний запрос в модель preprocessing.
3. Получаем предобработанное изображение «PREPROCESSED_OUTPUT» и мапим в «preprocess_image».
4. Посылаем «preprocess_image» в classification и segmentation модели.

Ансамблирование пайплайна

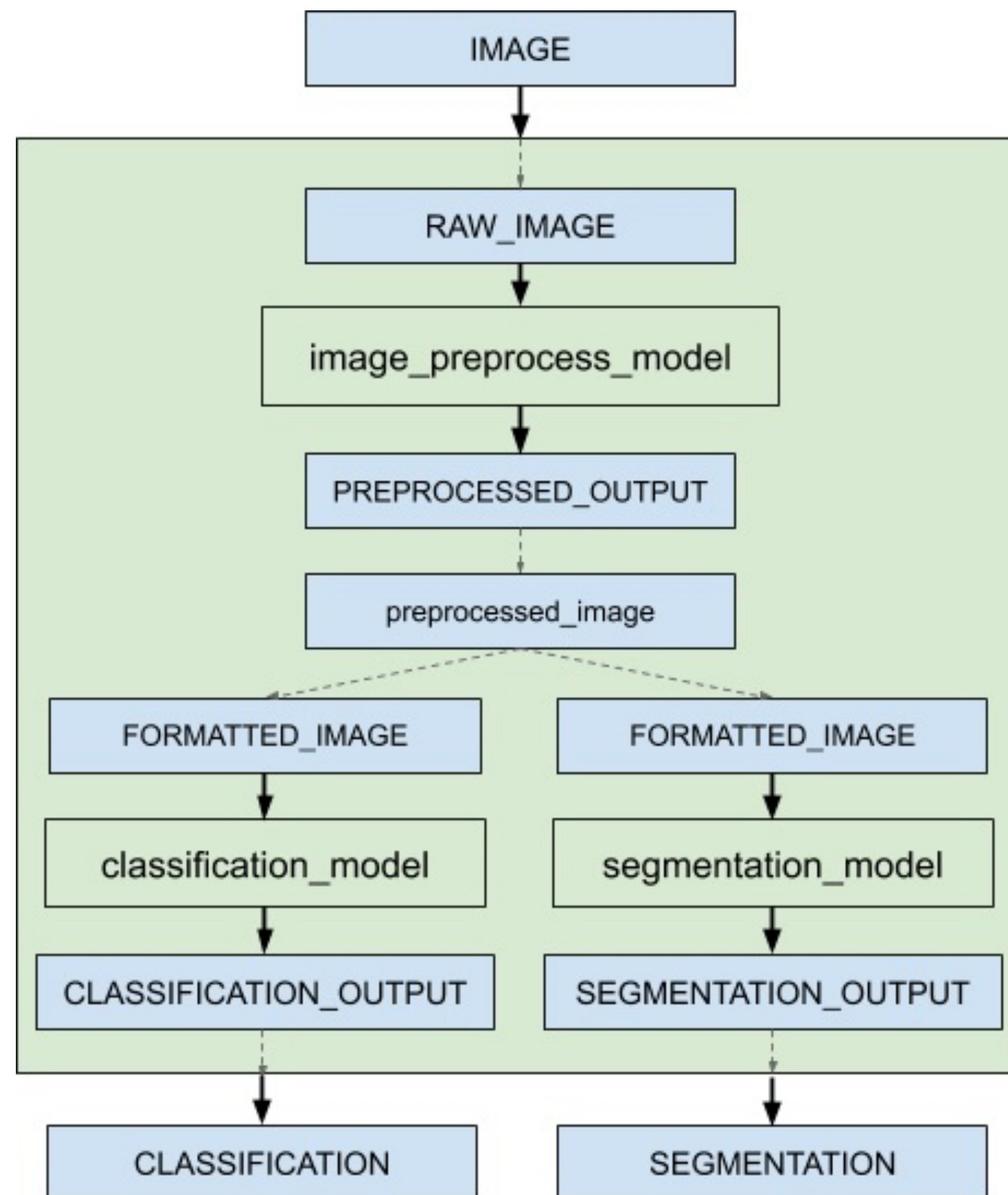
Ансамбль может иметь параллельную структуру.



1. На вход в ансамбль приходит изображение («IMAGE» tensor), которое сопоставляется input «RAW_IMAGE».
2. Отправляется внутренний запрос в модель preprocessing.
3. Получаем предобработанное изображение «PREPROCESSED_OUTPUT» и мапим в «preprocess_image».
4. Посылаем «preprocess_image» в classification и segmentation модели.
5. Получаем результаты инференса моделей, возвращаем их.

Ансамблирование пайплайна

Ансамбль может иметь параллельную структуру.



ozon{ech

```
ensemble_scheduling {
  step [
    {
      model_name: "image_preprocess_model"
      model_version: -1
      input_map {
        key: "RAW_IMAGE"
        value: "IMAGE"
      }
      output_map {
        key: "PREPROCESSED_OUTPUT"
        value: "preprocessed_image"
      }
    },
    {
      model_name: "classification_model"
      model_version: -1
      input_map {
        key: "FORMATTED_IMAGE"
        value: "preprocessed_image"
      }
      output_map {
        key: "CLASSIFICATION_OUTPUT"
        value: "CLASSIFICATION"
      }
    },
    {
      model_name: "segmentation_model"
      model_version: -1
      input_map {
        key: "FORMATTED_IMAGE"
        value: "preprocessed_image"
      }
      output_map {
        key: "SEGMENTATION_OUTPUT"
        value: "SEGMENTATION"
      }
    }
  ]
}
```

Ансамблирование пайплайна

Оптимизация ансамблей

В отличие от других моделей, **ансамбли не поддерживают параметр `instance_group`**, которым можно увеличивать число «копий» модели.

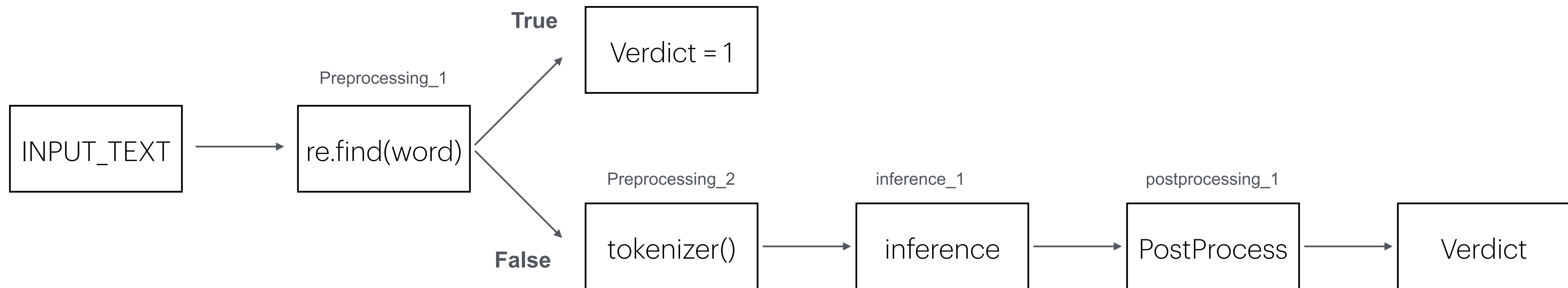
Причина в том, что сам **ансамбль в основном является событийно-управляемым планировщиком** с очень минимальными накладными расходами, поэтому он почти никогда не становится узким местом конвейера.

Составные модели в ансамбле могут быть индивидуально масштабированы по количеству с соответствующими настройками `instance_group`. Чтобы оптимизировать производительность ансамбля моделей, вы можете использовать **Model Analyzer** для поиска оптимальных конфигураций модели.

Ансамблирование пайплайна

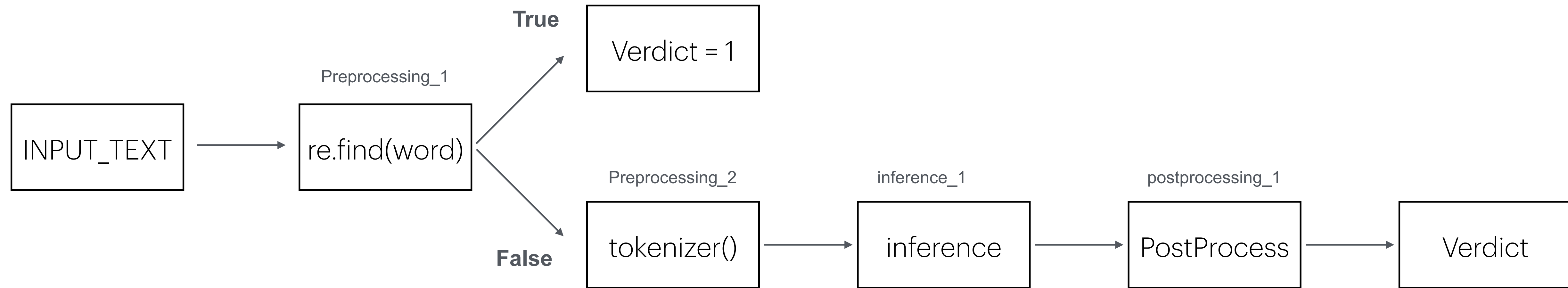
А что, если?

Ансамбль - **направленный граф без циклов**. А что делать, если в пайплайне используем более сложную логику if-else?



Ансамблирование пайплайна

А что, если?



BLS backend - позволяет выполнять запросы вывода на других моделях, обслуживаемых Triton, как часть выполнения вашей модели Python

BLS следует использовать только внутри функции `execute` и он не поддерживается в методах `initialize` или `finalize`

Ансамблирование пайплайна

А что, если?

BLS backend - позволяет выполнять запросы вывода на других моделях, обслуживаемых Triton, как часть выполнения вашей модели Python

Из основных фичей:

1. Писать разветвленные графы.
2. Выгружать и загружать модели для разовых операций.
3. Трекать реквесты внутри тритона и снимать расширенные метрики (кроме трейсов реквестов где-то из логов контейнера тритона).
4. Можно использовать асинхронность в пайплайне.

[Ссылка на документацию](#)

Ансамблирование пайплайна

Материалы

- Лекции от nvidia
- Пример nlp модели
- BLS