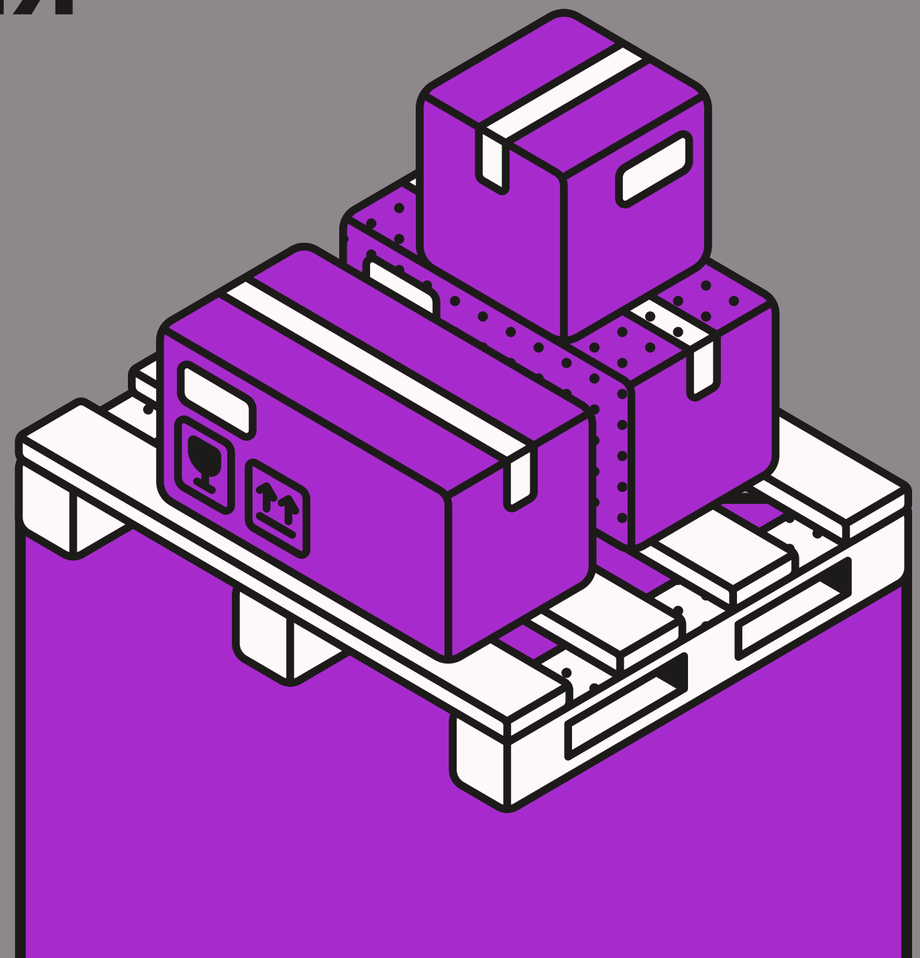
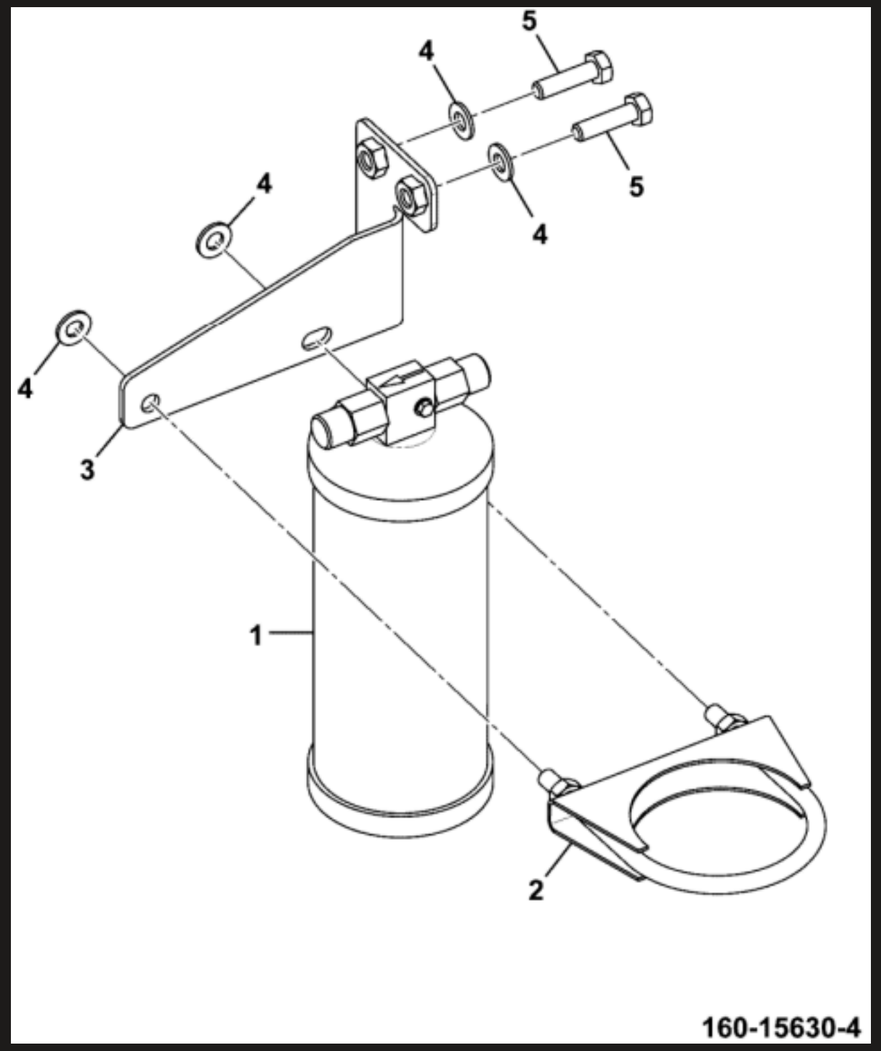
JCB PARTS

Поисковая система для запчастей JCB



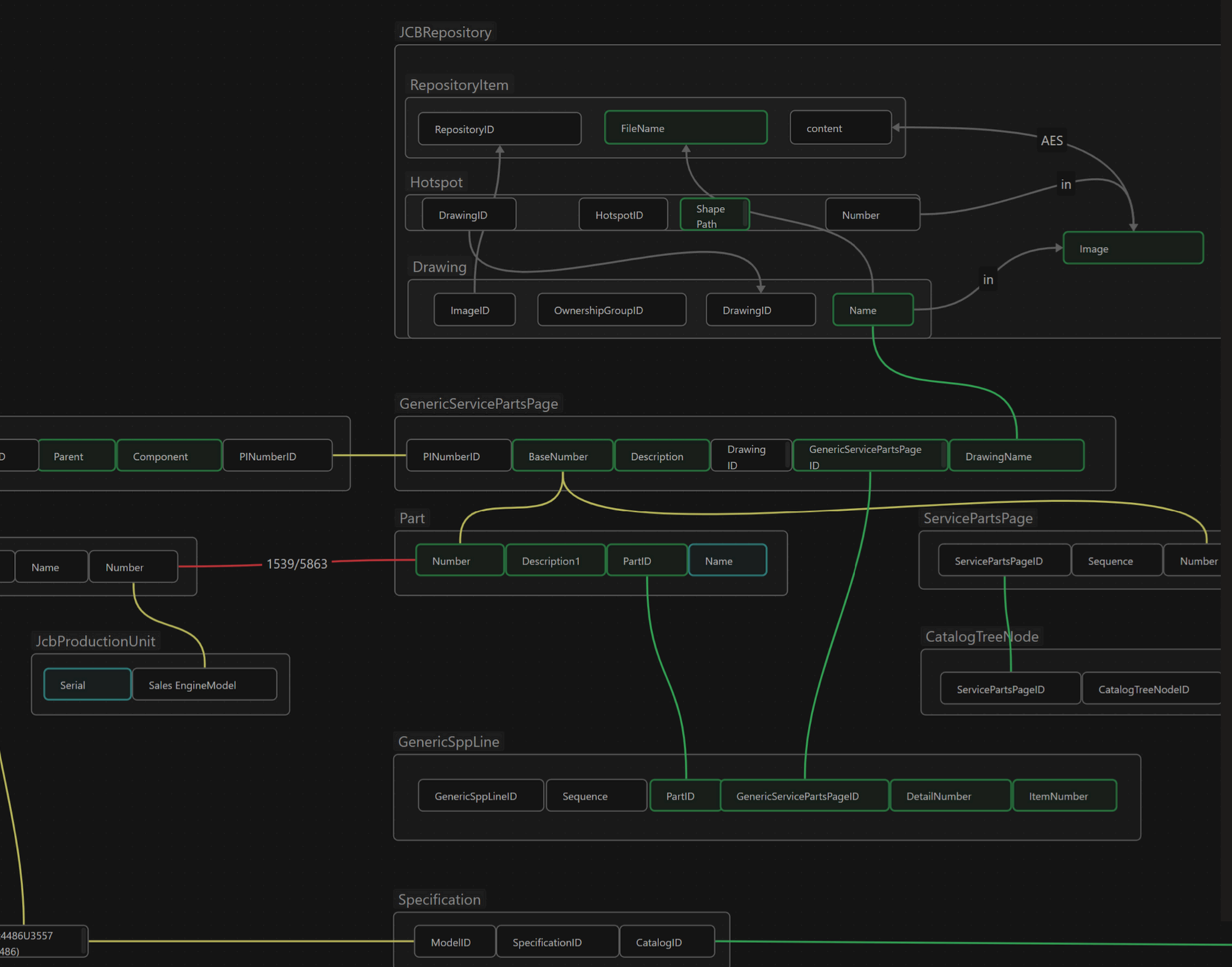
Let’s Move

Анализ изображений и чертежей



Конвертация текстовых изображений в файловый формат
Декодирование чертежей, зашифрованных через AES в формат TIFF

Анализ структуры изображений
Интерпретация данных из таблиц связанных с артикулом, деталью



Анализ базы данных

Выявление связей между таблицами через внешние ключи. Применение алгоритмов кластеризации для обнаружения повторов. Построение связей для получения полных данных о деталях: артикулы, описания, модели применения

Сбор и обработка данных

- Собраны данные категории, артикулы, названия, описания, характеристики, фото и цены с двух сайтов (jcbdetal.ru и q-parts.kz) по запчастям JCB.
  Проводился парсинг этих сайтов для того чтобы забрать русские метки.
- Разработан единый формат хранения данных с разбивкой по вкладкам и категориям.
- Выполнено сопоставление товаров по фото и названиям.
- Проведена проверка совпадений и точности переводов.
- Подготовлена финальная таблица (артикул, названия и описания на русском и английском).

https://jcbdetal.ru/katalog-zapshasti-jcb/
https://www.q-parts.kz/catalog/k-9835007-jcb

Переводчик

```
[ ]  training_args = Seq2SeqTrainingArguments(
        output_dir='./results_4run',
        eval_strategy='epoch',
        learning_rate=2e-5,
        per_device_train_batch_size=16,
        per_device_eval_batch_size=16,
        weight_decay=0.01,
        save_total_limit=1,
        num_train_epochs=10,
        predict_with_generate=True,
        fp16=True,
        report_to='none'
    )
```

[2610/2610 11:39, Epoch 10/10]

Epoch	Training Loss	Validation Loss	Chrf
1	No log	0.423607	73.744319
2	0.989200	0.183488	86.140861
3	0.989200	0.127202	88.006600
4	0.154300	0.102310	90.475667
5	0.154300	0.094910	89.622273
6	0.080200	0.081837	90.718900
7	0.080200	0.083324	90.271725
8	0.055100	0.076102	90.754749
9	0.055100	0.073181	92.107169
10	0.044300	0.071902	91.445367

Была проделана работа над pretrained "Helsinki-NLP/opus-mt-en-ru" eng2rus техническим переводчиком. Дообучен на наших данных.

Автоэнкодер

```python
class AE(nn.Module):
  def __init__(self, vocab_size, embedding_dim=EMB_DIM, hidden_dim=HID_DIM):
    super(AE, self).__init__()
    self.embedding = nn.Embedding(vocab_size, embedding_dim, padding_idx=0)
    self.encoder = nn.LSTM(embedding_dim, hidden_dim, batch_first=True)
    self.decoder = nn.LSTM(embedding_dim, hidden_dim, batch_first=True)
    self.output = nn.Linear(hidden_dim, vocab_size)


  def forward(self, x):
    embedded = self.embedding(x)
    _, (h_n, _) = self.encoder(embedded)

    decoder_input = torch.zeros_like(x)
    decoder_emb = self.embedding(decoder_input)

    decoded, _ = self.decoder(decoder_emb, (h_n, torch.zeros_like(h_n)))
    out = self.output(decoded)
    return out
```

Первая итерация AE, добились loss'a ~2

```python
class AE(nn.Module):
  def __init__(self, vocab_size, embedding_dim, hidden_dim):
    super(AE, self).__init__()
    self.embedding = nn.Embedding(vocab_size, embedding_dim, padding_idx=0)
    #self.dropout = nn.Dropout(0.3)
    self.encoder = nn.LSTM(embedding_dim, hidden_dim, batch_first=True, bidirectional=True)
    self.bridge = nn.Linear(hidden_dim * 2, hidden_dim)
    self.decoder = nn.LSTM(embedding_dim, hidden_dim, batch_first=True)
    self.output = nn.Linear(hidden_dim, vocab_size)

  def forward(self, x):
    embedded = self.embedding(x)
    _, (h_n, _) = self.encoder(embedded)
    h_cat = torch.cat((h_n[0], h_n[1]), dim=1)
    decoder_h0 = self.bridge(h_cat).unsqueeze(0)
    decoder_c0 = torch.zeros_like(decoder_h0)

    decoder_input = x[:, :-1]
    decoder_target = x[:, 1:]

    decoder_emb = self.embedding(decoder_input)
    decoded, _ = self.decoder(decoder_emb, (decoder_h0, decoder_c0))
    out = self.output(decoded)
    return out, decoder_target
'''
```

Двунаправленность LSTM
Подход, аналогичный BERT
- Маскирование случайных токенов во время обучения
- Предсказание замаскированных токенов как дополнительная задача

```
EPOCH 1/15 - LOSS: 1.7993 | CosineSim: similar=0.8706 | random=0.1138
EPOCH 2/15 - LOSS: 0.8441 | CosineSim: similar=0.8132 | random=0.0714
EPOCH 3/15 - LOSS: 0.7651 | CosineSim: similar=0.8043 | random=0.1359
EPOCH 4/15 - LOSS: 0.7389 | CosineSim: similar=0.7864 | random=0.1118
EPOCH 5/15 - LOSS: 0.7181 | CosineSim: similar=0.7794 | random=0.1030
EPOCH 6/15 - LOSS: 0.6948 | CosineSim: similar=0.7885 | random=0.1062
EPOCH 7/15 - LOSS: 0.6782 | CosineSim: similar=0.7941 | random=0.0958
EPOCH 8/15 - LOSS: 0.6652 | CosineSim: similar=0.7949 | random=0.1006
EPOCH 9/15 - LOSS: 0.6751 | CosineSim: similar=0.7987 | random=0.1042
EPOCH 10/15 - LOSS: 0.6746 | CosineSim: similar=0.7954 | random=0.0917
EPOCH 11/15 - LOSS: 0.6528 | CosineSim: similar=0.7969 | random=0.0968
EPOCH 12/15 - LOSS: 0.6545 | CosineSim: similar=0.8099 | random=0.0930
EPOCH 13/15 - LOSS: 0.6459 | CosineSim: similar=0.8010 | random=0.0910
EPOCH 14/15 - LOSS: 0.6510 | CosineSim: similar=0.8031 | random=0.1069
EPOCH 15/15 - LOSS: 0.6390 | CosineSim: similar=0.8015 | random=0.0985
```

Снижение функции потерь до ~0.7
Увеличение показателя cosine_margin до ~0.8

Поисковая система

```
def search(query, encoder, vocab, index, df, top_k=5)
  tokens = torch.tensor(encode(query, vocab), dtype=t

  with torch.no_grad():
    vec = encoder(tokens).squeeze(0).cpu().numpy
    D, I = index.search(vec, top_k)

    results = df.iloc[I[0]].copy()
    results['distance'] = D[0]
    return results
```

```
index.add(embeddings.astype('float32'))
id_to_function[index.ntotal - 1] = metadata
```

```
print(f"Number of vectors stored: {index.ntotal}")
print(f"Vector dimension: {index.d}")
print(f"Number of metadata entries: {len(id_to_function)
```

```
Number of vectors stored: 12146
Vector dimension: 512
Number of metadata entries: 1
```

FAISS (Facebook AI Similarity Search)

библиотека для эффективного поиска похожих векторов в больших коллекциях.

API-сервис

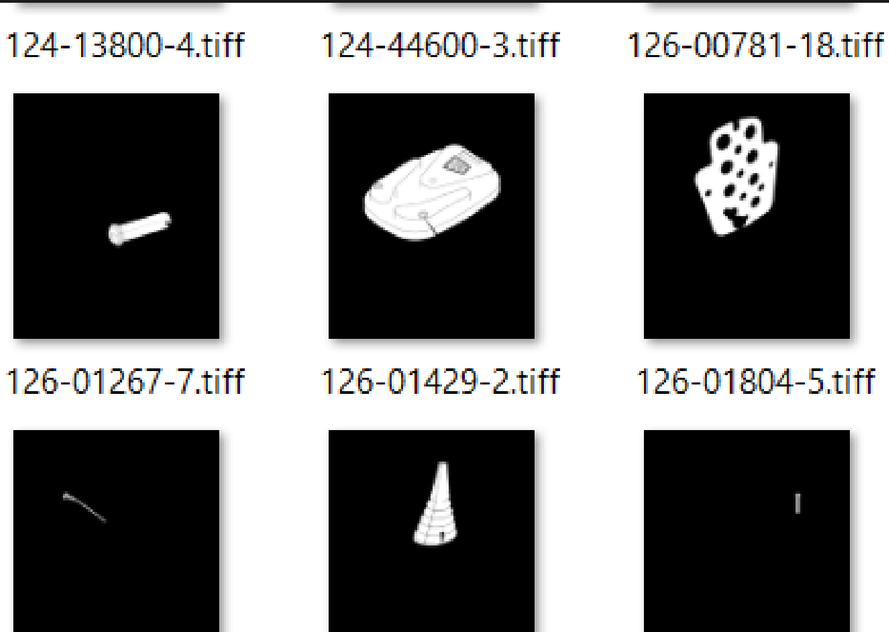
В настоящее время ведётся разработка и отладка API-сервиса, который объединит все компоненты системы и предоставит интерфейс для взаимодействия.

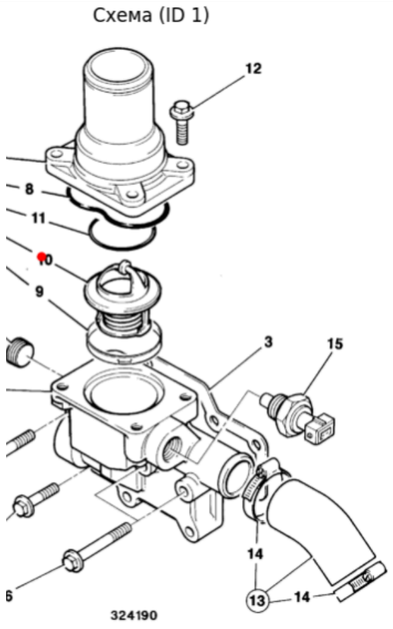
```python
class SearchQuery(BaseModel):
    text: str
    top_k: int = 10

class SearchResult(BaseModel):
    words2tokens: str
    distance: float

@app.get('/')
def home():
    return {'health_check': 'OK'}

@app.post("/search", response_model=List[SearchResult])
def search(query: SearchQuery):
    ids = torch.tensor([encode(query.text, vocab, MAX_LEN)], dtype=torch.long).to(DEVICE)
    with torch.no_grad():
        vec = encoder(ids).squeeze(0).cpu().numpy().astype("float32").reshape(1, -1)

    D, I = index.search(vec, query.top_k)
    results = df.iloc[I[0]][["words2tokens"]].copy()
    results["distance"] = D[0]
    return results.to_dict(orient="records")
```

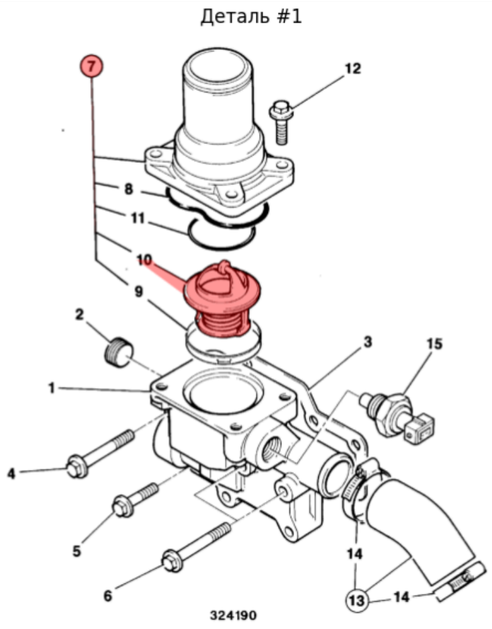
Отдельное направление деятельности составляет исследовательская работа (R&D) для разработки интеллектуального поискового механизма по фотографиям.
Исследование возможностей генеративно-состязательных сетей (GAN) для автоматической стилизации изображений на основе чертежей, аугментации цветных фотографии деталей на основе схем.

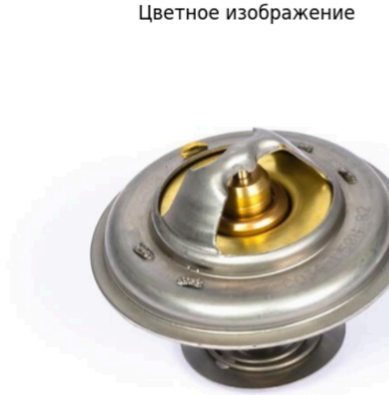
План создания масок для деталей на чертежах

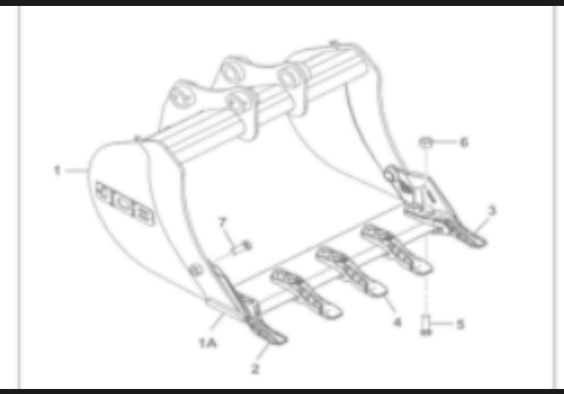
1. База для обучения: разметка 200+ масок вручную через Supervisely
2. Модель: адаптация SAM с использованием координат номеров деталей
3. Обучение: стандартное разделение 80/20 с аугментациями
4. Генерация масок: применение SAM с постобработкой
5. Контроль качества: проверка 8% результатов
6. Анализ ошибок: классификация проблемных случаев
7. Дообучение: фокус на сложных примерах с Focal Loss
8. Финализация: повторная генерация и сглаживание
9. Подготовка для GAN: создание пар "чертёж-фото" с нормализацией

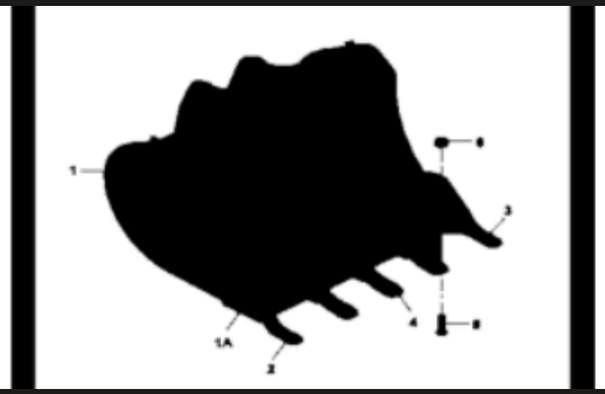












```python
class ConditionalPix2PixGenerator(nn.Module):
    def __init__(self, num_parts, embed_dim=128):
        super().__init__()
        self.embedding = nn.Embedding(num_parts, embed_dim)
        self.linear = nn.Linear(embed_dim, 512*512)
        self.encoder = nn.Sequential(
            nn.Conv2d(5, 64, 4, 2, 1), nn.ReLU(),
            nn.Conv2d(64, 128, 4, 2, 1), nn.BatchNorm2d(128), nn.ReLU(),
            nn.Conv2d(128, 256, 4, 2, 1), nn.BatchNorm2d(256), nn.ReLU()
        )
        self.decoder = nn.Sequential(
            nn.ConvTranspose2d(256, 128, 4, 2, 1), nn.BatchNorm2d(128), nn.ReLU(),
            nn.ConvTranspose2d(128, 64, 4, 2, 1), nn.BatchNorm2d(64), nn.ReLU(),
            nn.ConvTranspose2d(64, 3, 4, 2, 1), nn.Tanh()
        )

    def forward(self, input_tensor, part_id):
        b, c, h, w = input_tensor.shape
        part_emb = self.embedding(part_id)
        cond = self.linear(part_emb).view(b, 1, h, w)
        x = torch.cat([input_tensor, cond], dim=1)
        encoded = self.encoder(x)
        return self.decoder(encoded)

class PatchDiscriminator(nn.Module):
    def __init__(self):
        super().__init__()
        self.model = nn.Sequential(
            nn.Conv2d(7, 64, 4, 2, 1), nn.LeakyReLU(0.2),
            nn.Conv2d(64, 128, 4, 2, 1), nn.BatchNorm2d(128), nn.LeakyReLU(0.2),
            nn.Conv2d(128, 256, 4, 2, 1), nn.BatchNorm2d(256), nn.LeakyReLU(0.2),
            nn.Conv2d(256, 1, 4, 1, 1)
        )
```

```python
def evaluate_mask_similarity(mask, mask_idx, original_sketch_gray, ref_image_gray, ref_mask, contours_ref):
    if not np.any(mask):
        return None
    coords = cv2.findNonZero(mask)
    if coords is None:
        return None
    x, y, w, h = cv2.boundingRect(coords)
    schematic_region = original_sketch_gray[y:y+h, x:x+w]
    if schematic_region.size == 0:
        return None
    try:
        mask_region = mask[y:y+h, x:x+w].copy()
        contours_mask, _ = cv2.findContours(mask_region, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
        if contours_mask and contours_ref:
            cnt_m = max(contours_mask, key=cv2.contourArea)
            cnt_r = max(contours_ref,    key=cv2.contourArea)
            shape_similarity = compute_shape_similarity(cnt_m, cnt_r)
        else:
            shape_similarity = 0.0
        best_template_score = 0
        h_mask, w_mask = mask_region.shape
        h_ref, w_ref = ref_image_gray.shape
        scales = [0.5, 0.75, 1.0, 1.25, 1.5, 2.0]
        angles = [0, 30, 60, 90, -30, -60]
```