



TAPe+ML

User Guide

Computer Vision Platform for Detection,
Classification, and Segmentation

Document version: 1.0
Developer: Comexp Research Lab
Date: September 2026



Contents

1. About TApe+ML
2. Getting Started
3. Object Detection
 - 3.1. What Detection Does
 - 3.2. Running Detection
 - 3.3. Video Detection
 - 3.4. Models
4. Detection Sensitivity
5. Model Validation
 - 5.1. Understanding the Metrics
6. Inference Results
7. Segmentation
 - 7.1. What Segmentation Does
 - 7.2. Viewing Results
8. Classification
 - 8.1. What Classification Does
9. Custom Models
10. Fine-tuning
 - 10.1. Dataset Preparation
11. Annotation
12. Auto-Annotation
13. Working with Datasets
14. Hardware Emulation
15. CPU Performance
16. REST API
 - 16.1. Video Detection API
17. API Response
18. Benchmarks
19. Model Size
20. Plans and Access
21. Recommended Workflow
22. Practical Dataset Recommendations
23. Troubleshooting Common Issues
24. Technical Specifications
25. Performance and Limitations
26. Intellectual Property
27. Support
28. Quick Start

TAPe+ML

User Guide

Computer Vision Platform for Detection, Classification, and Segmentation

Document version: 1.0

Developer: Comexp Research Lab

Date: September 2026

1. About TAPe+ML

TAPe+ML is a Computer Vision platform for using pre-trained models, creating custom models from user data, evaluating model performance, and integrating Computer Vision capabilities into third-party applications through an API.

TAPe+ML can be used in different deployment scenarios, including **Cloud, On-Premise, and Edge**.

The platform supports both images and video streams and provides three core Computer Vision tasks:

- **Detection** — object detection;
- **Segmentation** — pixel-level object segmentation;
- **Classification** — image classification.

Data, models, testing, and API access are managed through a unified web interface.

TAPe+ML uses a proprietary architecture designed for compact models and efficient inference.

The model contains **fewer than 100,000 parameters**.

Key capabilities

TAPe+ML allows you to:

- run pre-trained models on images;
- perform real-time object detection on video streams;
- detect objects and identify their classes;
- generate pixel-precise segmentation masks;
- classify images;
- create custom models for new object classes;
- annotate images directly in the platform;
- use automatic annotation;
- validate models on a dedicated validation dataset;
- view inference results;
- receive real-time video detection events;
- evaluate inference performance;
- emulate performance on different CPU and Edge devices;
- integrate models through REST API.

The web interface at **ml.comexp.net** is intended for the Cloud version of the platform. Local and Edge deployments are available for relevant projects and are arranged with Comexp.

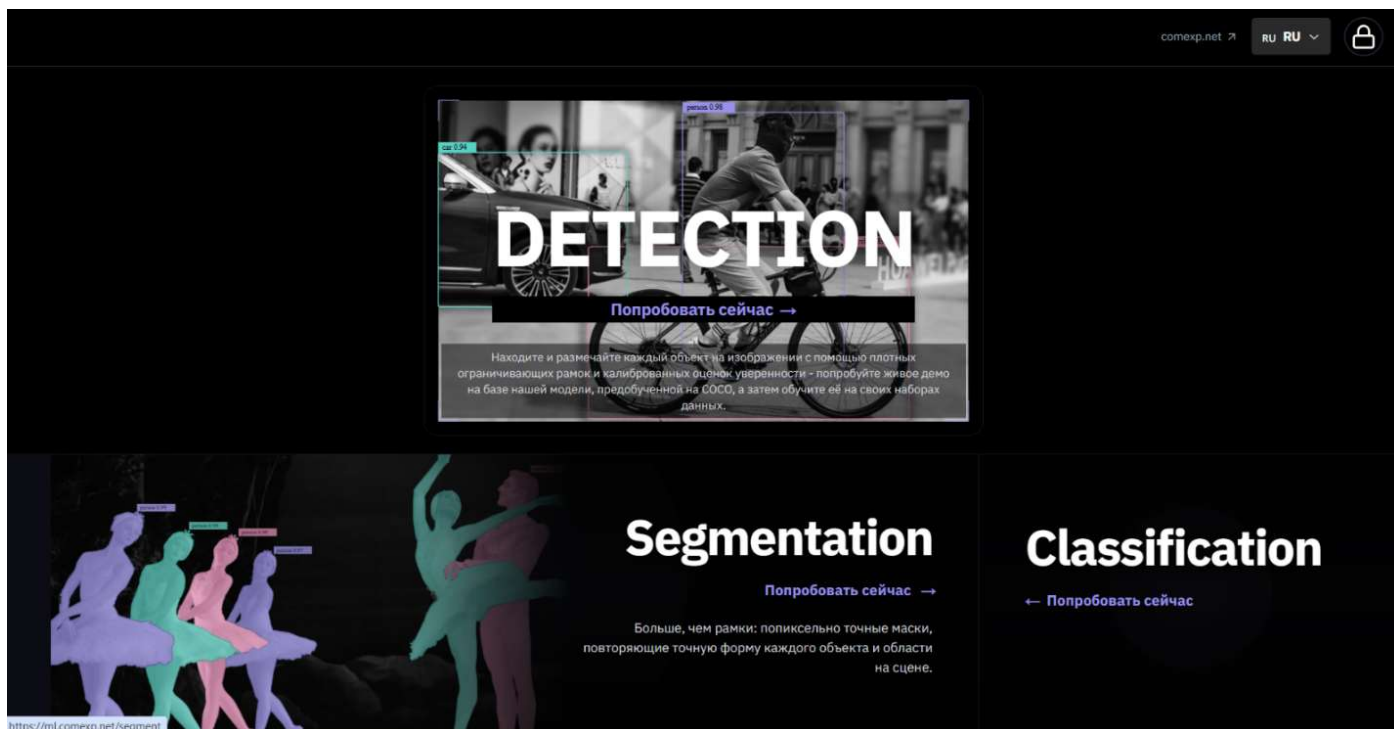


Figure 1. TAPe+ML main page and available Computer Vision tasks.

2. Getting Started

Working with TAPe+ML starts with selecting the required Computer Vision task.

The home page provides three main modes.

Detection

Find objects in an image using bounding boxes and confidence scores.

Segmentation

Detect objects and generate pixel-precise masks.

Classification

Identify the most likely image classes together with confidence scores.

Select the appropriate section depending on your task.

3. Object Detection

3.1. What Detection Does

Object Detection allows you to upload an image and identify the objects contained in it.

For each detected object, the system returns:

- object class;
- confidence score;
- object coordinates;
- inference time.

The base model is pre-trained on **COCO** and supports 80 object classes.

If you need to detect objects that are not included in the base model, you can create a custom model with your own classes.

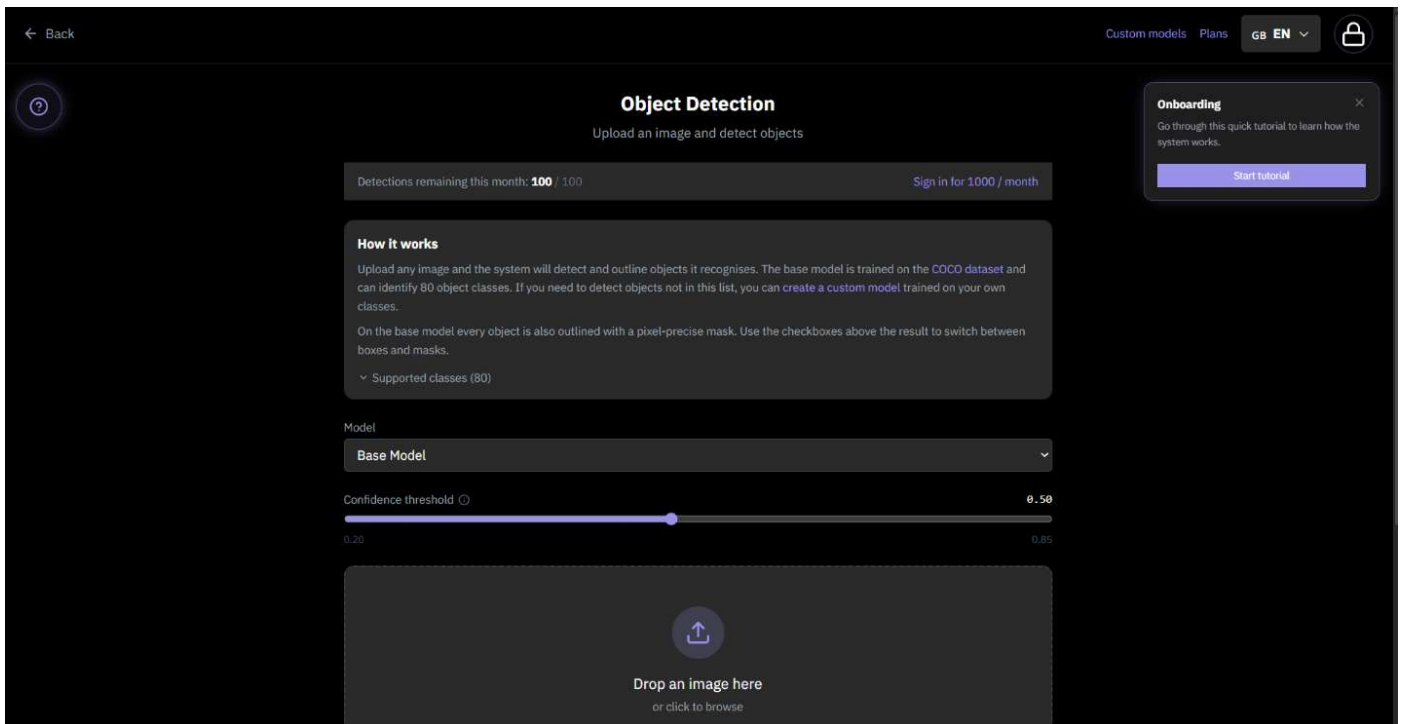


Figure 2. Object Detection: image upload and model selection.

3.2. Running Detection

To run object detection:

1. Open **Object Detection**.
2. Select a model.
3. Set the required **confidence threshold**.
4. Upload an image.
5. Start inference.
6. Review the detected objects and inference time.

Supported image formats:

- JPG;
- PNG;
- WebP.

Maximum file size: **10 MB**.

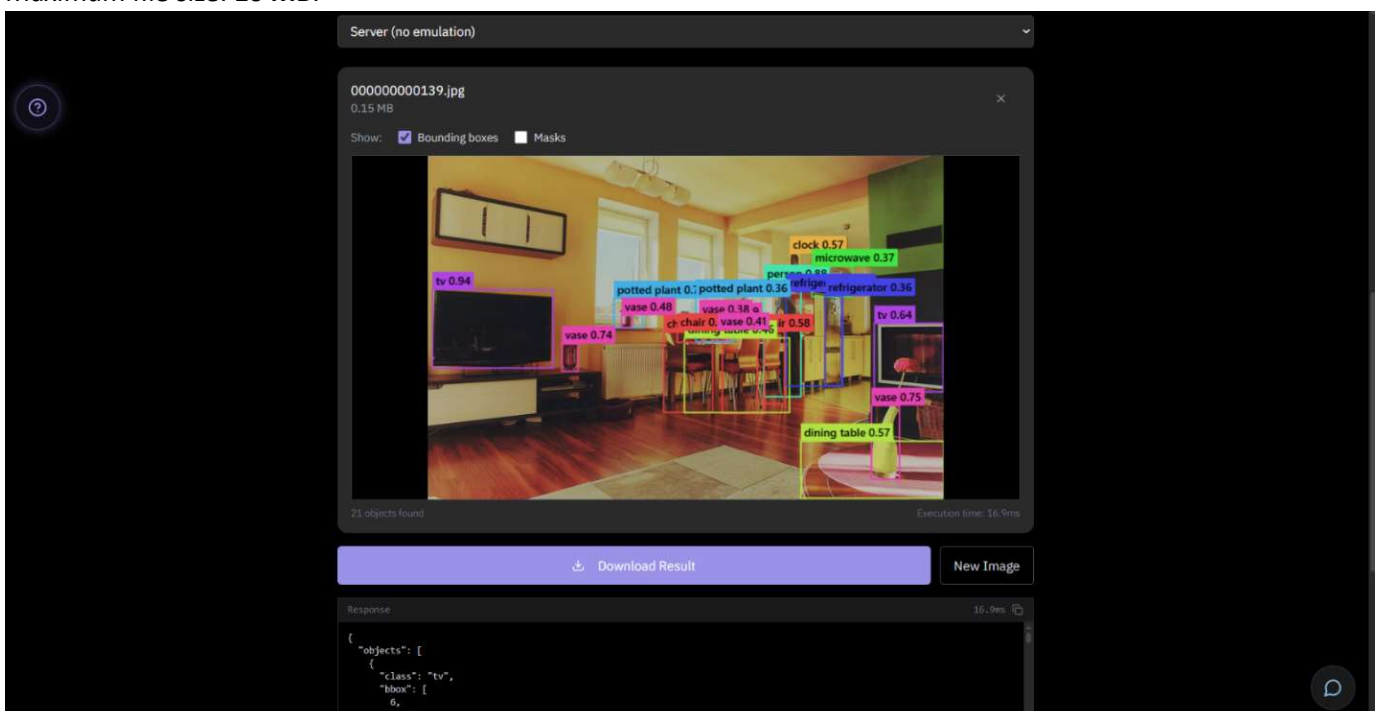


Figure 3. Detection result showing object classes and confidence scores.

3.3. Video Detection

TAPe+ML supports real-time object detection on video streams.

Two standard connection methods are available:

- **SRT** — the user sends a video stream to TAPe+ML;
- **RTSP** — TAPe+ML connects directly to the user's video stream.

After the connection is established, the system processes incoming frames and returns detection results in real time.

SRT

You can use any FFmpeg build with SRT support to send video.

The current SRT endpoint and connection parameters are displayed directly in the TAPe+ML interface.

RTSP

TAPe+ML can connect directly to an RTSP stream from an IP camera or another video source.

Enter the RTSP URL in the corresponding field.

Detection Results

Video processing results are available through **Server-Sent Events (SSE)**.

A JSON event can be returned for each processed frame and may contain:

- bounding box coordinates in pixels;
- object class;
- confidence score;
- other result parameters.

The number of frames submitted for processing is controlled by the video source on the user's side. In the current configuration, TAPe+ML does not impose a separate fixed FPS limit.

Video Detection is available starting with the **Med** plan.

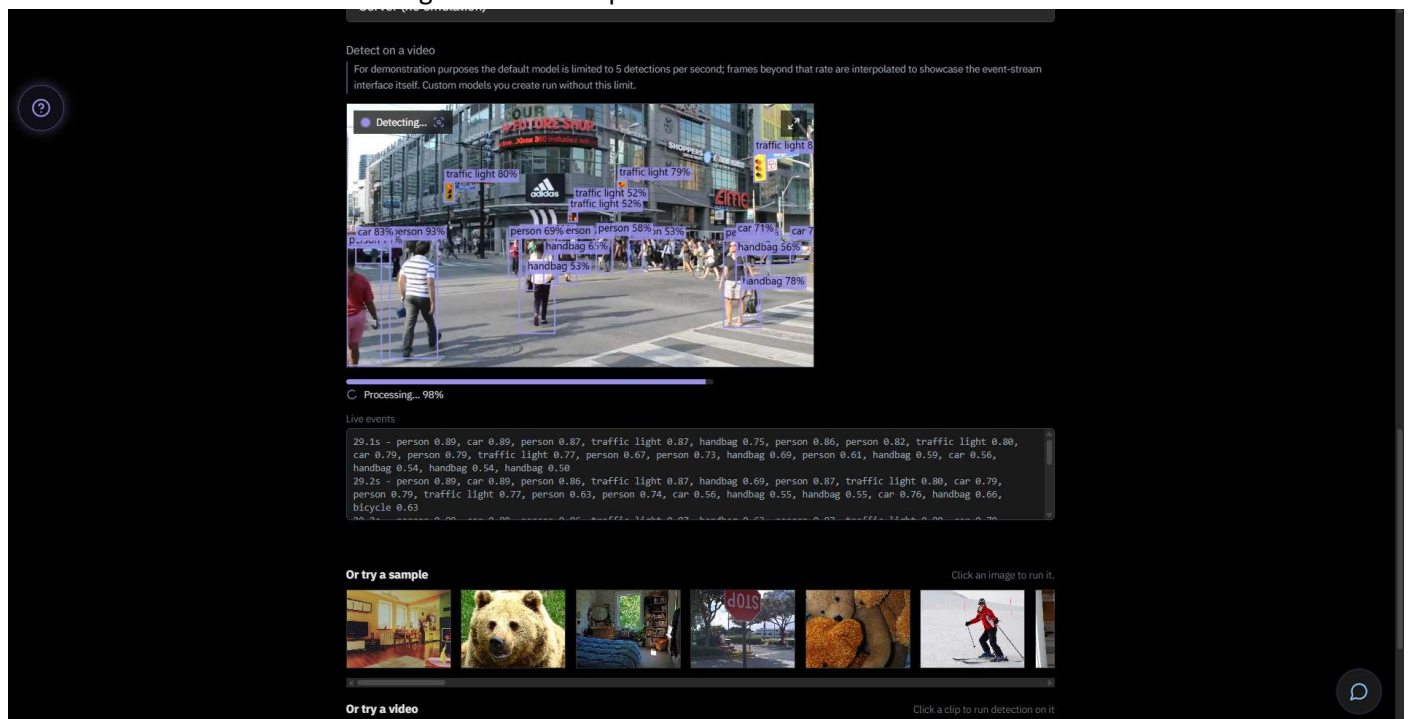


Figure 4. Video Detection result showing object classes and confidence scores.

3.4. Models

In the **Models** section, users can select pre-trained TAPe+ML demonstration models or custom models trained on their own data.

A separate **Our Models** section provides additional models that can be used to explore the effect of different configuration settings.

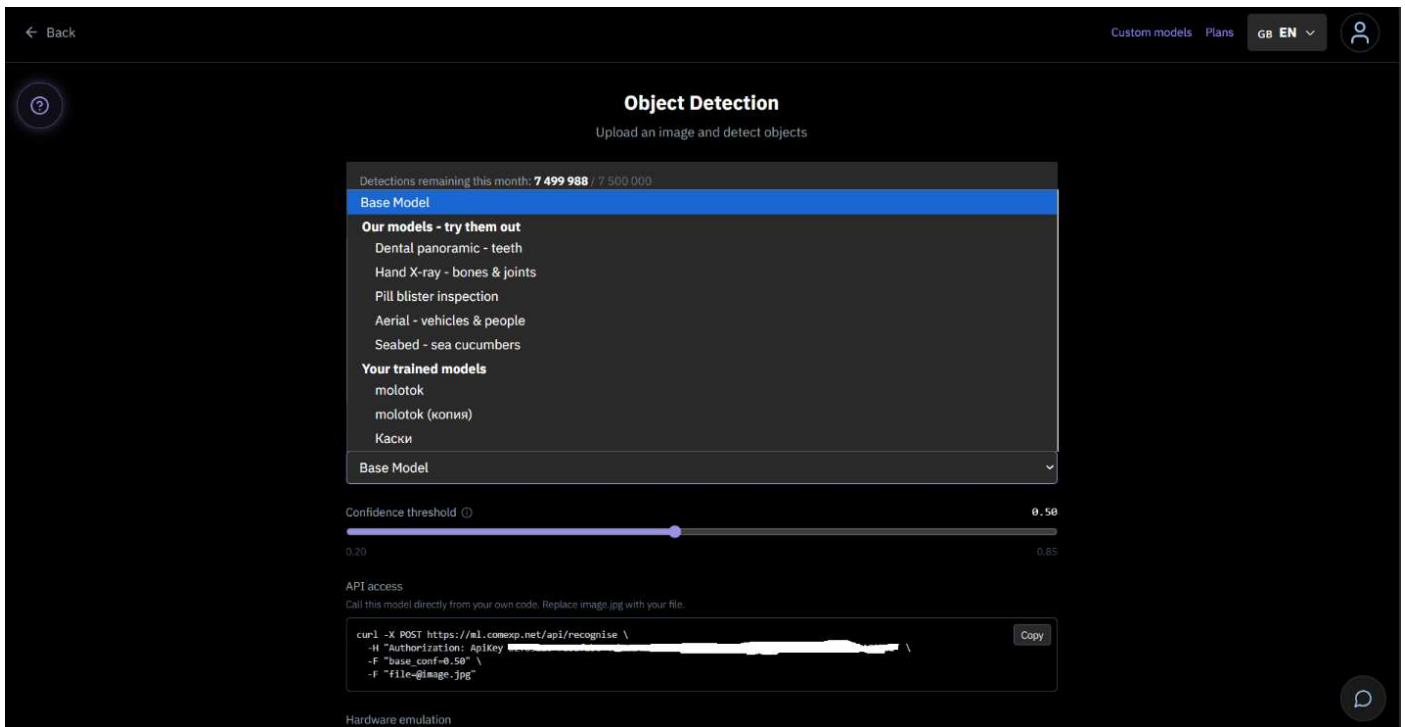


Figure 5. TApe+ML model management and selection.

4. Detection Sensitivity

The detection threshold can be adjusted using **Detection Sensitivity / confidence threshold**.

For example:

- 0.20;
- 0.30;
- 0.40;
- 0.50;
- and other values.

A lower threshold allows the model to detect more potential objects, but may increase the number of false positives. A higher threshold makes the model more selective and can reduce false positives, but may also cause some objects to be missed.

The appropriate threshold should therefore be selected using your own data and production requirements.

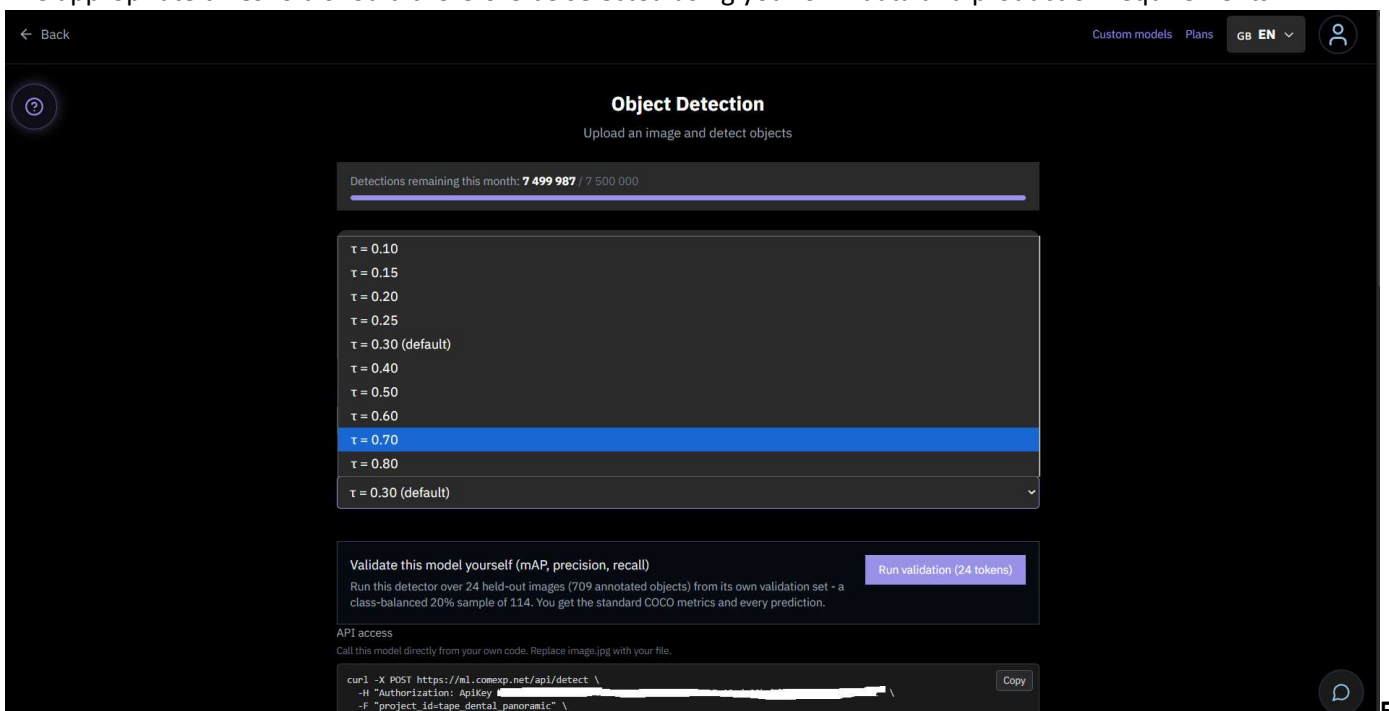


Figure 6. Detection Sensitivity settings and confidence threshold selection.

5. Model Validation

TAPe+ML allows you to validate a model separately using a validation dataset.

The interface provides the following function:

Validate this model yourself

The main object detection metrics available for model evaluation are:

- **mAP50;**
- **mAP50-95;**
- **Precision;**
- **Recall.**

For some models, validation is performed on a pre-selected portion of the validation dataset.

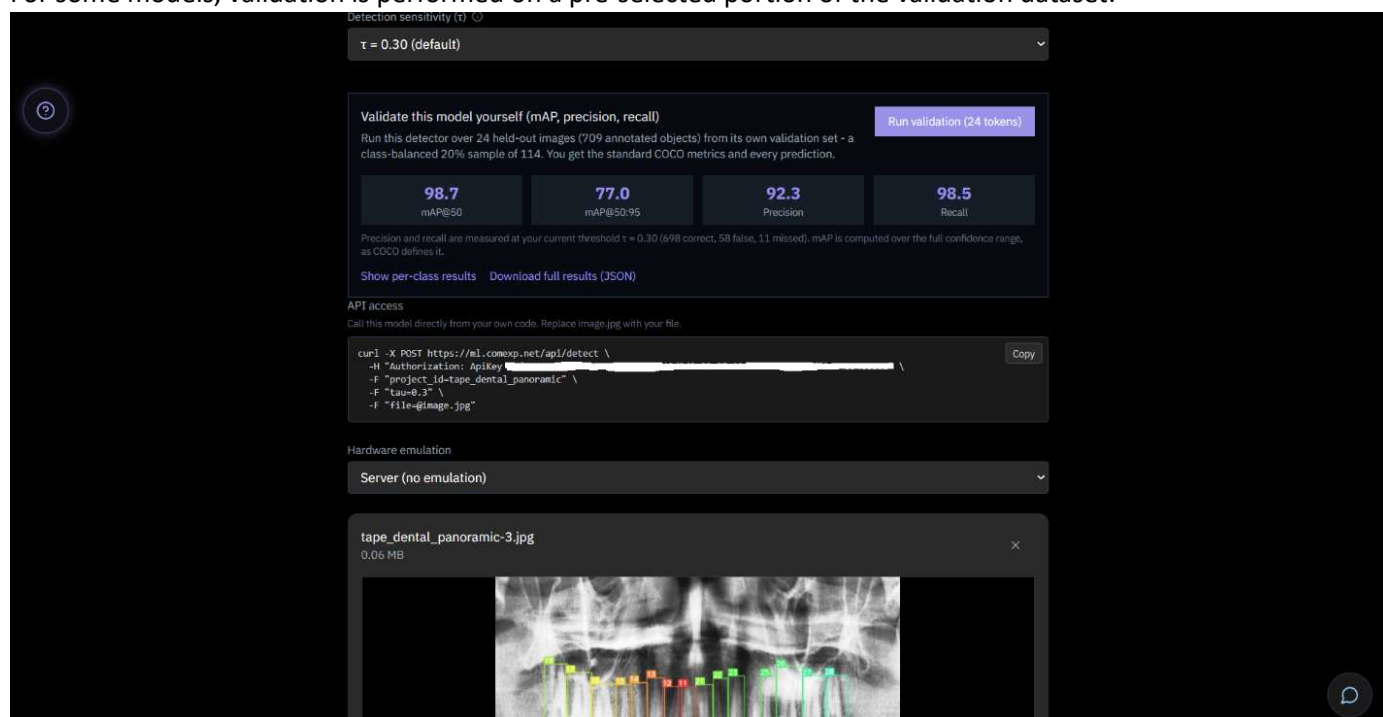


Figure 7. Built-in model validation showing mAP, Precision, Recall, and prediction results.

5.1. Understanding the Metrics

mAP50

Mean Average Precision calculated at an IoU threshold of 0.50.

mAP50-95

A stricter metric calculated across IoU thresholds from 0.50 to 0.95.

Precision

The proportion of correct detections among all detections produced by the model.

Recall

The proportion of existing objects that the model successfully detects.

For a specific production scenario, it is recommended to evaluate not only a single metric, but also Precision and Recall at the selected confidence threshold.

6. Inference Results

After an image has been processed, the result is displayed directly in the interface.

Depending on the task, the user may see:

- detected objects;
- classes;
- confidence scores;
- bounding boxes;
- segmentation masks;
- inference time.

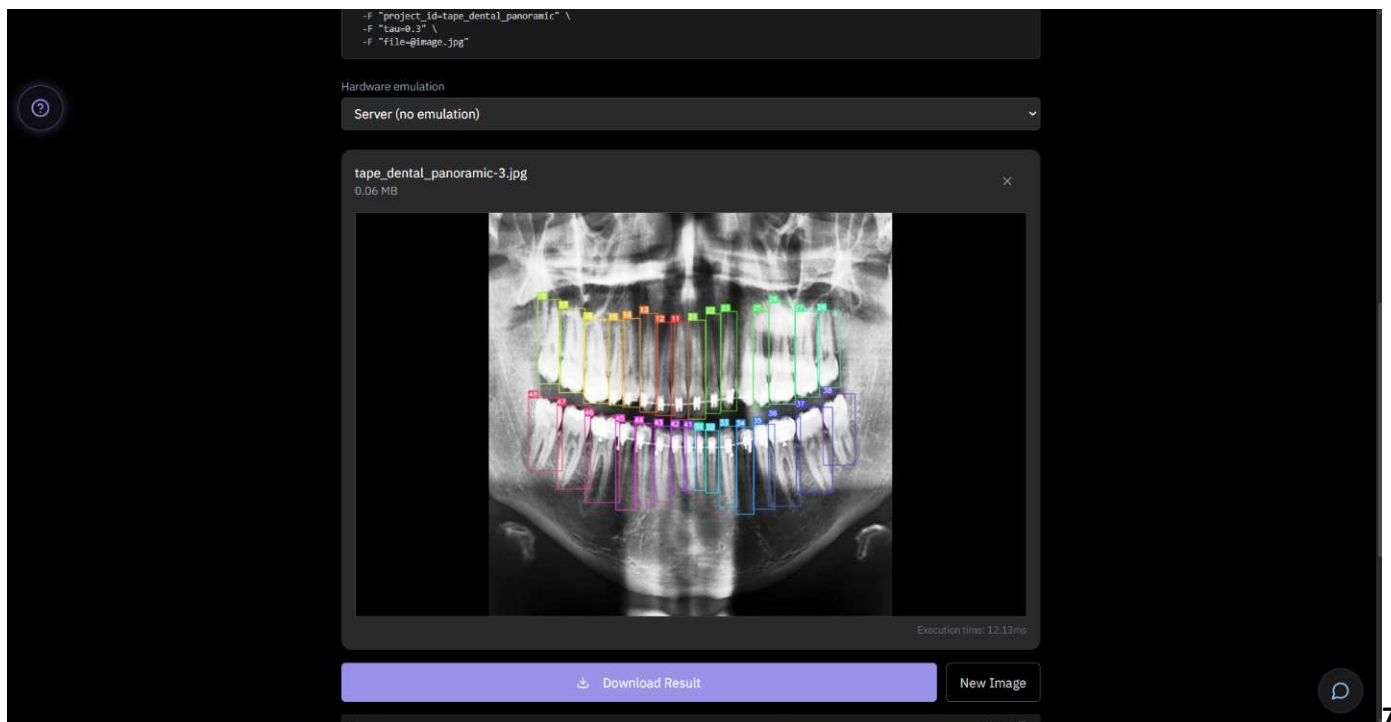


Figure 8. Inference result and processing time.

7. Segmentation

7.1. What Segmentation Does

Segmentation determines not only the location of an object, but also its shape at the pixel level.

For each detected object, the model generates a **segmentation mask** describing the area occupied by that object.

This is useful when a rectangular bounding box is not sufficient for the task.

For example:

- determining the exact shape of an object;
- measuring object area;
- identifying object boundaries;
- distinguishing multiple objects belonging to the same class.

The base segmentation model is pre-trained on COCO and supports 80 classes.

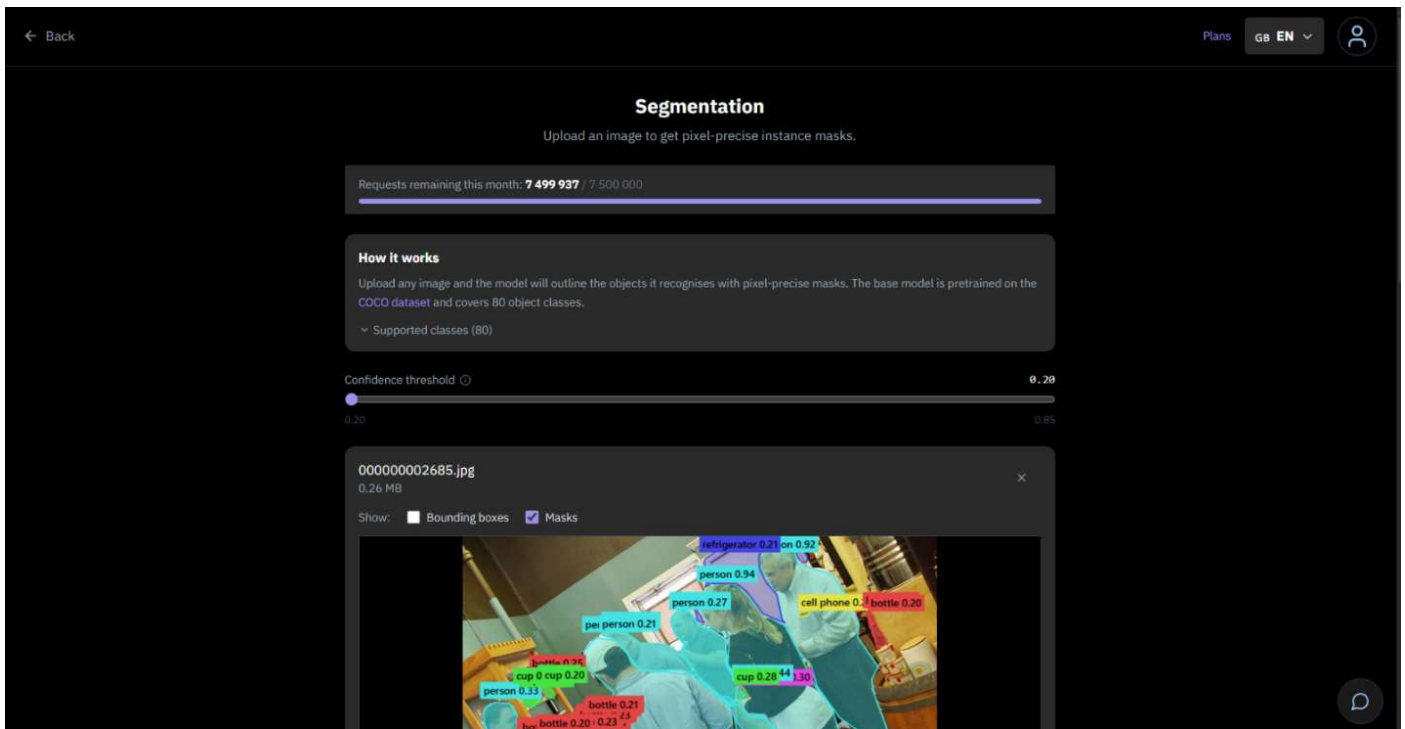


Figure 9. Segmentation section: generating pixel-precise instance masks

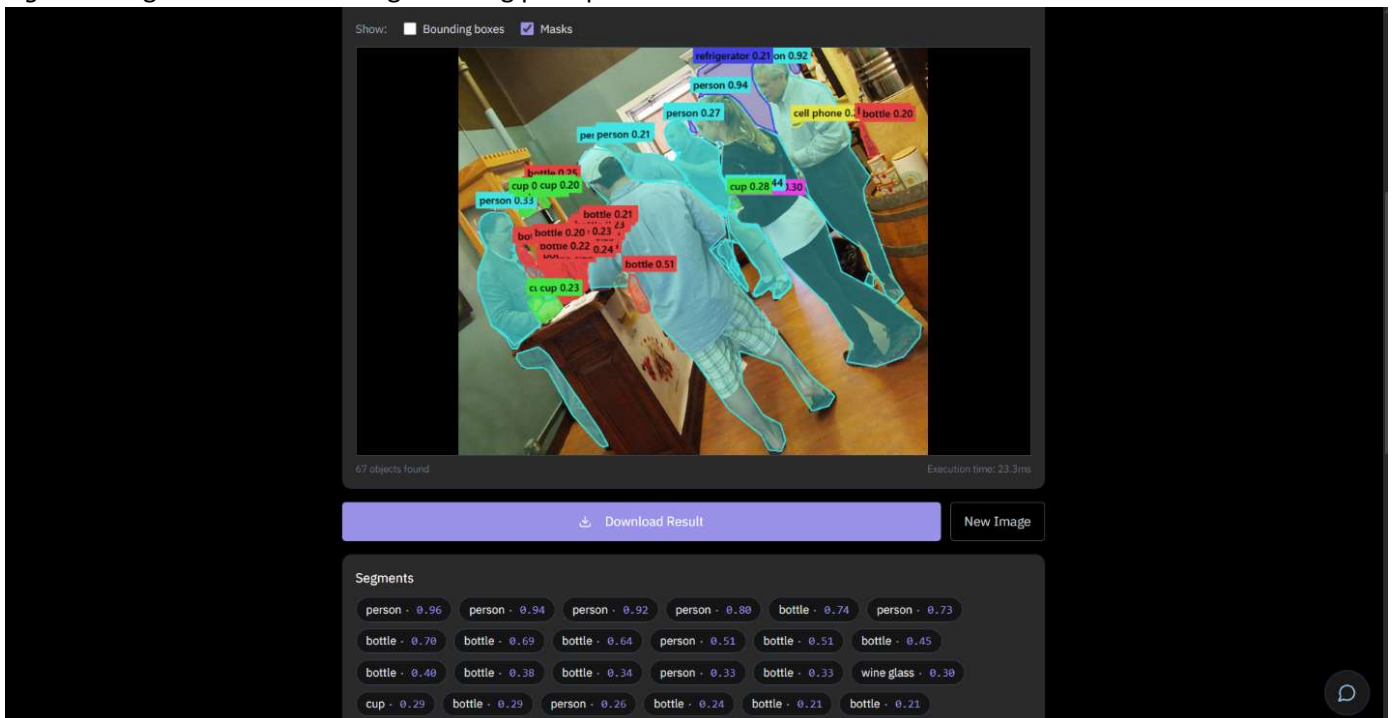


Figure 10. Segmentation result showing bounding boxes and pixel-level masks.

7.2. Viewing Results

The interface allows you to switch between:

- **Bounding Boxes;**
- **Masks.**

For each detected object, the class and confidence score are displayed.

8. Classification

8.1. What Classification Does

Classification identifies the most likely classes for an image.

After an image is uploaded, the system returns:

- the most likely class;
- confidence score;
- additional classification results.

Classification is suitable for tasks where the goal is to determine the category of an image or object.

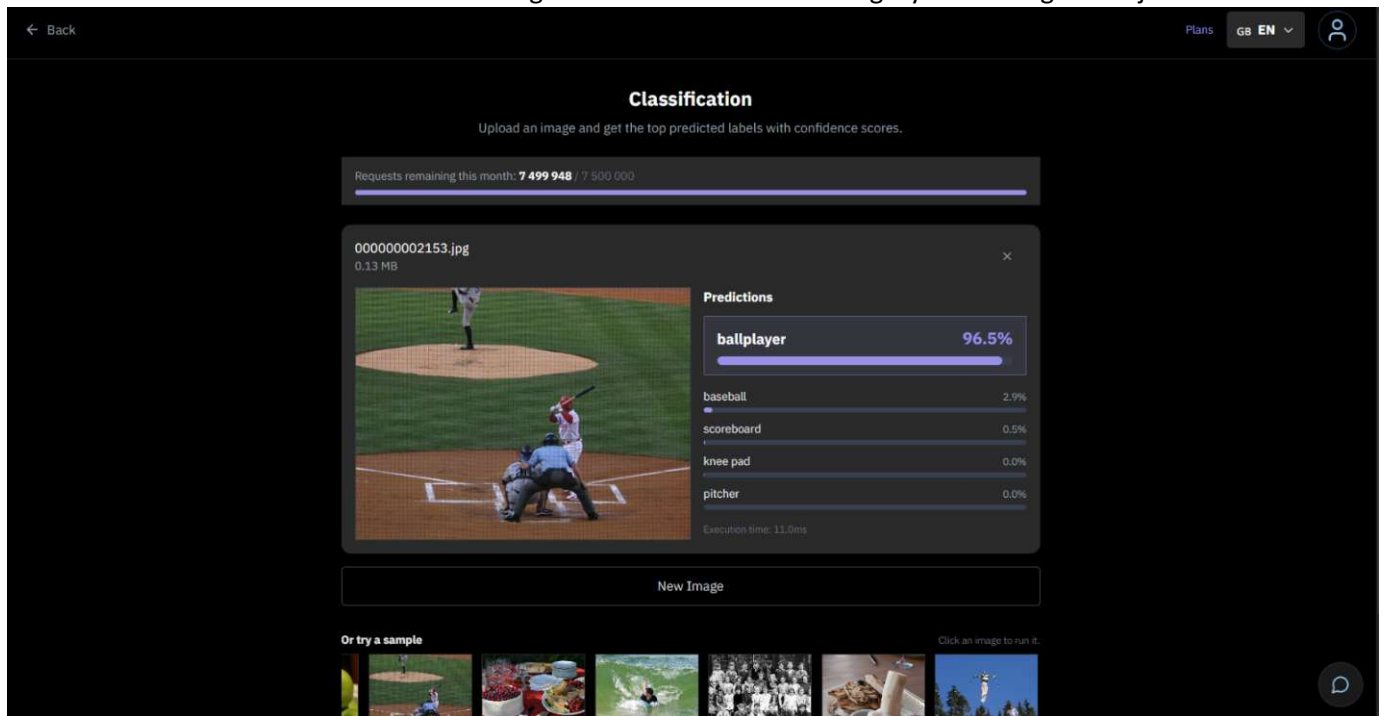


Figure 11. Classification result with confidence scores.

9. Custom Models

TAPe+ML allows you to create models for custom object classes.

For example, the base model may know the **person** class, while your application needs to detect a specific type of equipment, defect, package, or medical object.

In this case, you can create a custom model trained for your specific classes.

10. Fine-tuning

The **Fine-tuning** section is used to create custom detection models.

The overall workflow is:

Dataset → Annotation → Fine-tuning → Validation → Custom Model → Detection

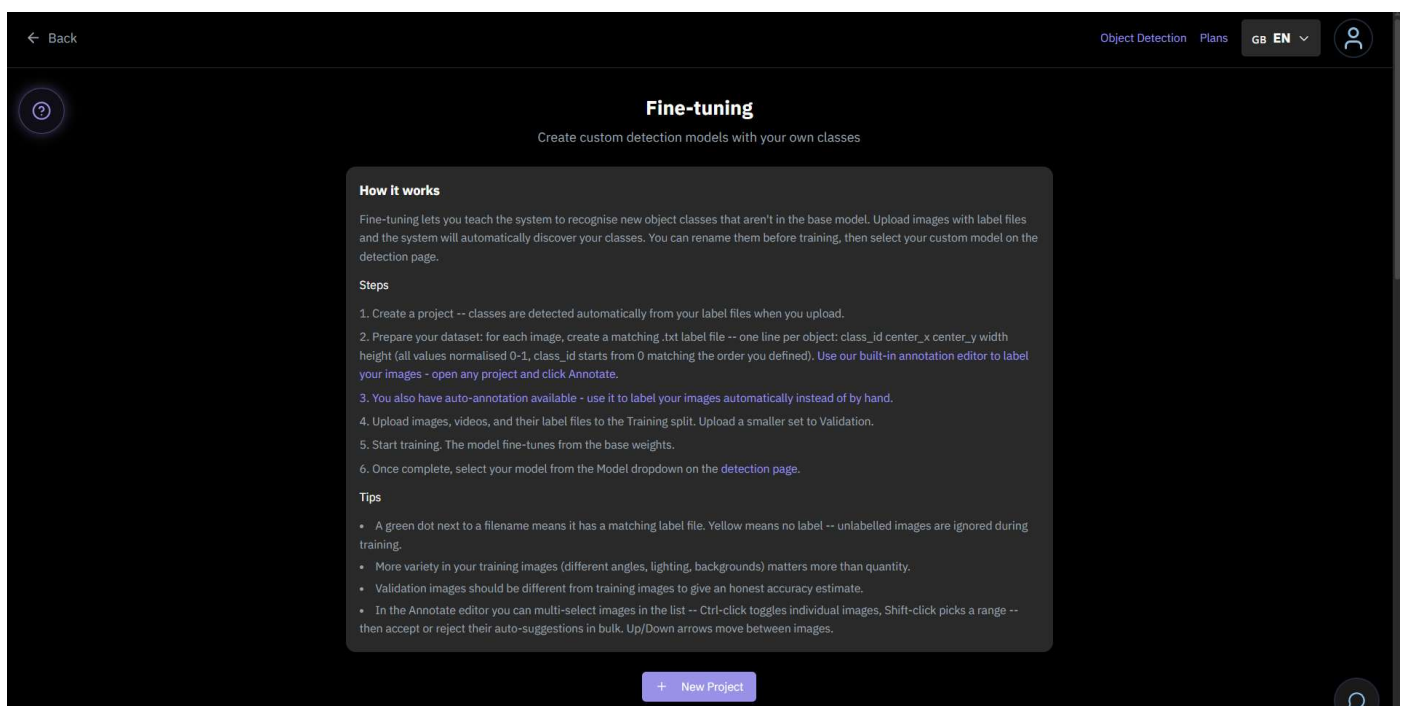


Figure 12 Creating a custom model using Fine-tuning

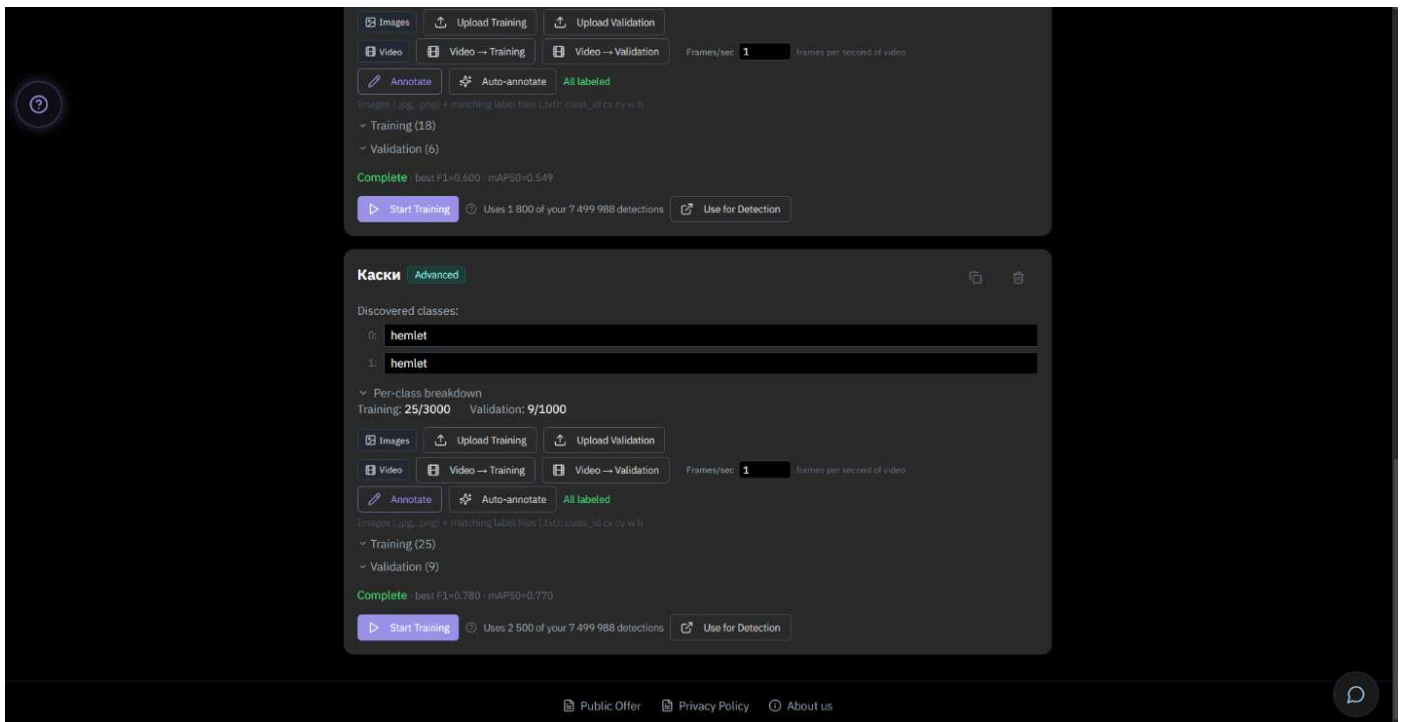


Figure 13 Custom model training settings

10.1. Dataset Preparation

To train a detection model, you need:

- images;
- corresponding label files.

Each image must have a corresponding .txt label file.

Each line in a label file describes one object:

class_id center_x center_y width height

Coordinates are normalized to the **0–1** range.

class_id starts at **0**.

11. Annotation

TAPe+ML includes a built-in annotation editor.

This allows you to perform annotation directly inside the platform without requiring a separate annotation tool.

You can:

- open an image;
- create bounding boxes;
- assign classes;
- review existing annotations;
- use auto-annotation;
- accept or reject automatically suggested annotations.

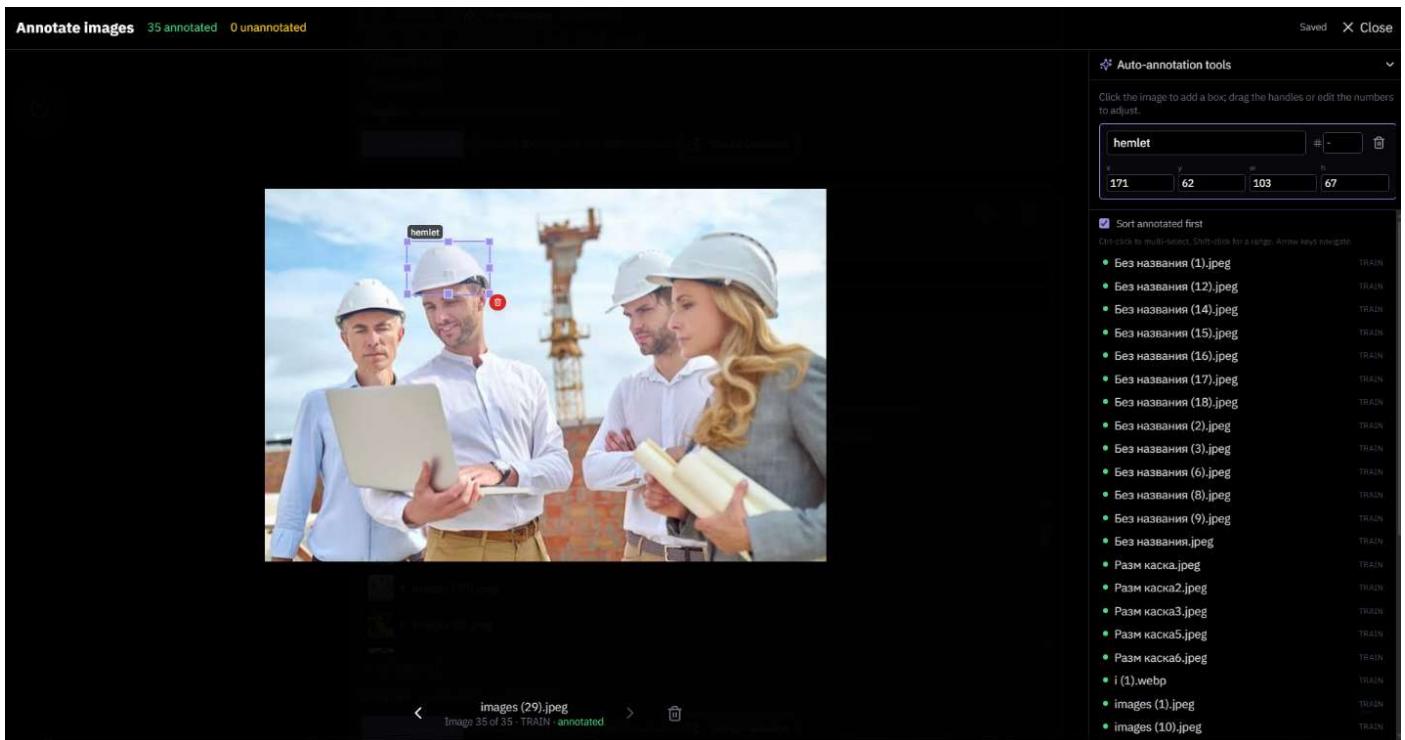


Figure 14. Built-in TAPe+ML annotation editor.

12. Auto-Annotation

Auto-annotation can be used to speed up dataset preparation.

A typical workflow is:

1. Manually annotate an initial set of images.
2. Train or adapt a model.
3. Run the model on additional images.
4. Generate automatic annotation suggestions.
5. Review the suggestions.
6. Accept or reject them.
7. Use the verified dataset for further training.

For suitable tasks, the initial sample may consist of approximately **10–15 manually annotated images**.

The actual amount of data required depends on the complexity of the specific task.

13. Working with Datasets

The Annotation interface allows you to work with multiple images simultaneously.

Bulk selection operations are available to speed up annotation:

- **Ctrl + click** — select individual images;
- **Shift + click** — select a range;
- **Up/Down** — navigate between images;
- bulk accept or reject auto-annotation suggestions.

14. Hardware Emulation

One of the features of TAPe+ML is the ability to evaluate model performance using **Hardware Emulation**.

The interface provides emulation options for different hardware configurations, including:

- Server;
- Radxa Rock v2.0;
- Raspberry Pi 5 (8 GB);

- Raspberry Pi 4 (4 GB);
- RK3588 SBC;
- N5095 mini-PC;
- i3-N305 mini-PC.

This allows you to estimate how a model may perform on different hardware configurations before deploying it to the target device.

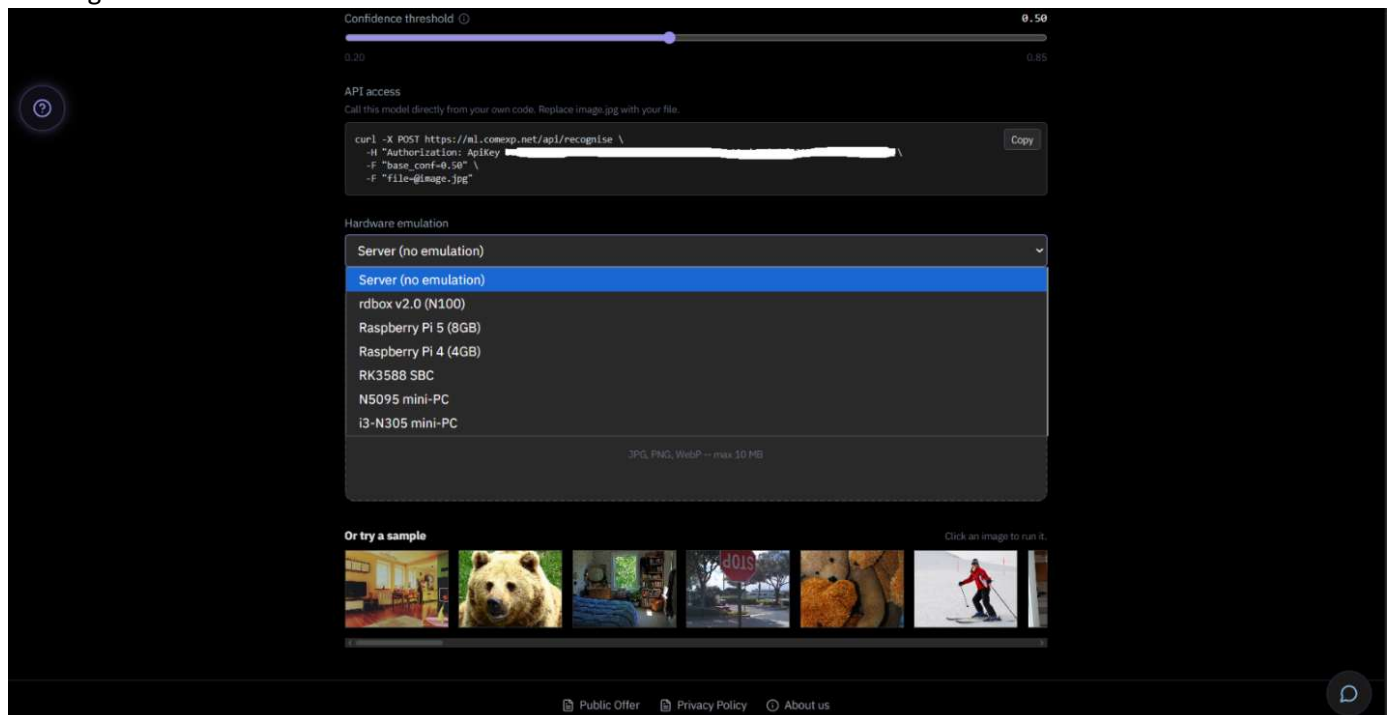


Figure 15. Hardware configuration selection for performance emulation.

15. CPU Performance

TAPe+ML is designed for efficient inference without requiring a GPU.

Reference measurements were performed using the following configuration:

Parameter	Configuration
CPU	Intel Xeon E5-2697 v2 @ 2.70 GHz
Processing mode	1 CPU core + SIMD
RAM	256 GB DDR3
Memory speed	1600 MT/s

Model inference takes approximately **8 ms per frame**.

The full pipeline, including inference and subsequent result processing, may take approximately **15 ms**, depending on the task and configuration.

Actual performance depends on:

- CPU;
- image resolution;
- preprocessing;
- postprocessing;
- number of detected objects;
- specific model;
- software configuration.

Performance on the actual production hardware should therefore be evaluated directly on the target platform.

16. REST API

TAPe+ML provides a **REST API** for integrating models with user applications.

For each model, the interface displays a current API request example, including the endpoint, request parameters, and image transmission format.

Depending on the model type and API mode, the corresponding endpoint shown directly in the product interface should be used.

An API request may include:

- image;
- project or model ID;
- confidence threshold;
- additional parameters supported by the specific model.

16.1. Video Detection API

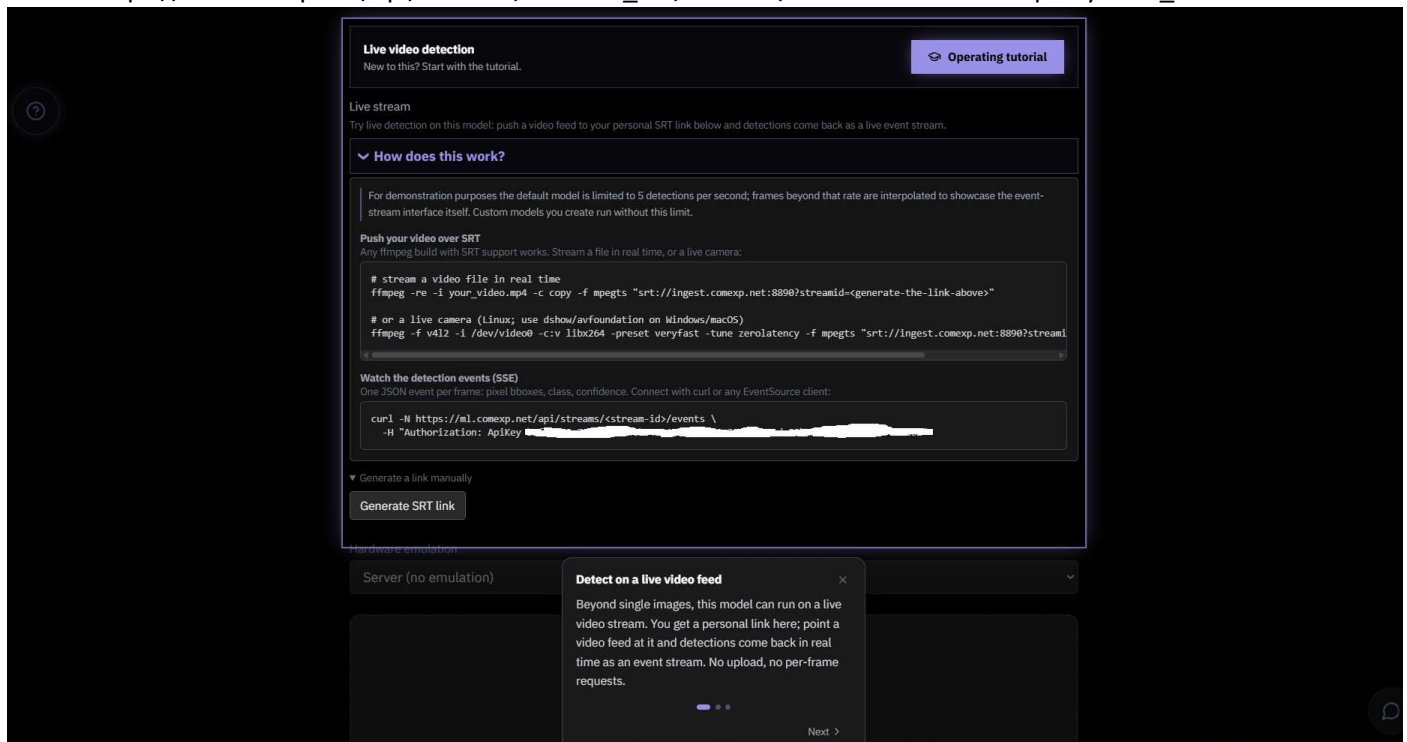
For Video Detection, TAPe+ML provides a stream of events through **Server-Sent Events (SSE)**.

SSE allows applications to receive detection results as video frames are processed, without repeatedly polling the server for new results.

A separate JSON event may be returned for each processed frame.

Example connection

```
curl -N https://ml.comexp.net/api/streams/<stream_id>/events \ -H "Authorization: ApiKey <API_KEY>"
```



The specific stream_id, endpoint, and authorization parameters depend on the connected video stream and are displayed in the TAPe+ML interface.

Figure 15. Video Detection settings.

17. API Response

After an API request is processed, the system returns a structured result.

The response may contain:

- detected objects;
- confidence scores;
- coordinates;
- segmentation polygons/masks for supported models;
- other result parameters;
- processing time.

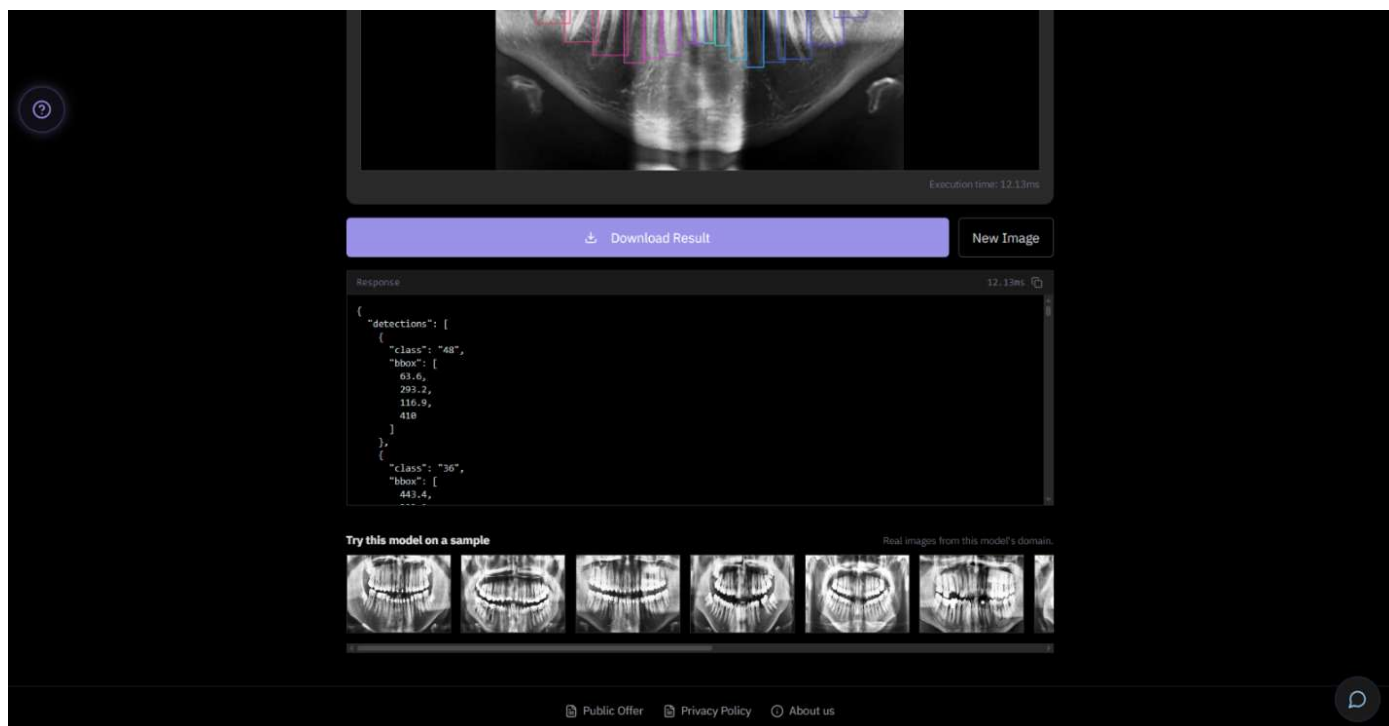


Figure 16. Example API response and inference time.

18. Benchmarks

The following results were obtained on public benchmark datasets.

COCO Object Detection

Metric	Result
mAP50	85.2%
mAP50-95	65.3%

COCO Instance Segmentation

Metric	Result
Mask AP50	80.7%
Mask AP50-95	58.4%

RF100-VL

Metric	Result
mAP50-95	68.3%

Benchmark results represent measurements obtained under specific conditions and correspond to particular model versions and evaluation methodologies.

Actual performance on a user's dataset may differ.

19. Model Size

TAPe+ML models contain **fewer than 100,000 parameters**.

The exact number of parameters may vary slightly depending on the dataset, task, configuration, and local inference version.

The model remains below **100,000 parameters**.

The compact model size makes TAPe+ML suitable for:

- CPU inference;
- Edge AI;
- embedded systems;
- local processing;
- environments with limited computational resources.

20. Plans and Access

TAPe+ML provides different access plans.

Available capabilities may include:

- inference request limits;
- access to base models;
- custom model training;
- Advanced TAPe+ML models;
- Batch API;
- auto-annotation;
- advanced training parameters.

Current pricing and plan features are displayed directly in the **Plans & Access** section and may change over time.

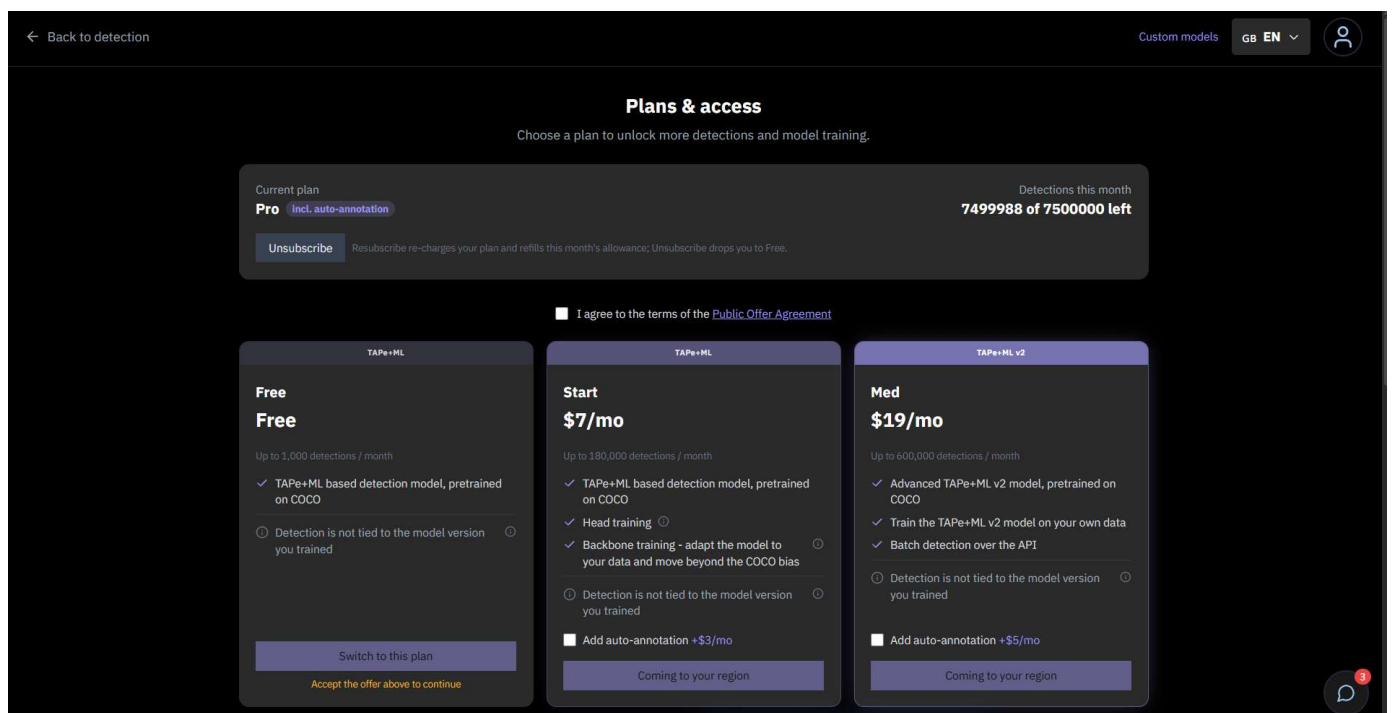


Figure 17. Available TAPe+ML plan.

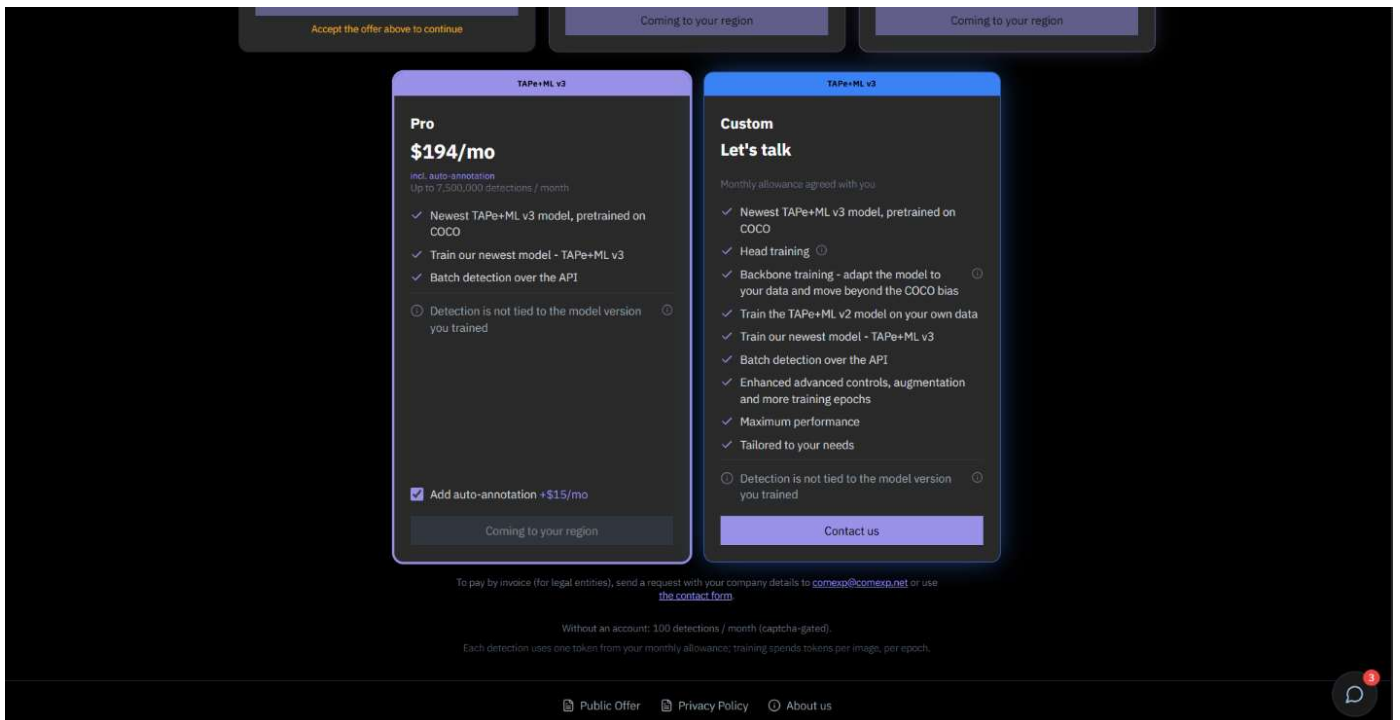


Figure 18. Plan features and configuration options.

21. Recommended Workflow

For a new Computer Vision project, the following workflow is recommended.

Step 1

Select the required task:

Detection / Segmentation / Classification

Step 2

Test the base model using your own images or video stream.

Step 3

If the base model does not contain the required class, create a **Custom Model**.

Step 4

Prepare the Dataset.

Step 5

Perform annotation using the built-in **Annotation** tool.

Step 6

Use **Auto-annotation** if appropriate.

Step 7

Split the data into **Training** and **Validation** datasets.

Step 8

Run **Fine-tuning**.

Step 9

Validate the model using **Validate**.

Step 10

Adjust **Detection Sensitivity**.

Step 11

Test inference on new images.

Step 12

Evaluate performance using **Hardware Emulation** or directly on the target hardware.

Step 13

Integrate the model through the **REST API**.

22. Practical Dataset Recommendations

For stable performance, use images that are as representative as possible of real production conditions.

The Dataset should ideally include:

- different camera angles;
- different distances to the object;
- different lighting conditions;
- different backgrounds;
- different object sizes;
- partial occlusions;
- visual variations of the target object.

Data diversity is often more important than simply increasing the number of images.

The **Validation Dataset** should be different from the **Training Dataset**.

A recommended split is **80/20 (Training Dataset / Validation Dataset)**.

This provides a more objective estimate of how well the model performs on previously unseen data.

23. Troubleshooting Common Issues

The model misses objects

Try the following:

- add more diverse examples;
- add images containing small objects;
- add images with partial occlusions;
- check the confidence threshold;
- check annotation quality.

Too many false positives

Try:

- increasing the threshold;
- adding negative examples;
- adding images containing visually similar objects;
- checking annotation correctness.

Low Precision

Review false positives and add examples of situations that the model incorrectly identifies as the target object.

Low Recall

Review missed objects and add examples representing the corresponding conditions.

Low inference speed

Check:

- target CPU;
- image resolution;
- preprocessing;
- postprocessing;
- network connection;
- selected model.

Use **Hardware Emulation** to preliminarily compare different hardware configurations.

24. Technical Specifications

Specification	TAPe+ML
Architecture	Proprietary TAPe+ML architecture
Model parameters	<100,000
Detection	Yes
Classification	Yes
Instance Segmentation	Yes
CPU inference	Yes
GPU required for inference	No
Annotation	Yes
Auto-annotation	Yes
Custom Models	Yes
Fine-tuning	Yes
Validation	Yes
Hardware Emulation	Yes
REST API	Yes
Video Detection	Yes
SRT video input	Yes
RTSP video input	Yes
Real-time detection events	Yes
SSE	Yes
Cloud	Yes
On-Premise	Available on request
Edge	Supported

25. Performance and Limitations

All published performance and accuracy figures are **reference results**.

Actual results depend on:

- model;
- Dataset;
- number of classes;
- annotation quality;
- image resolution;
- CPU;
- inference configuration;
- preprocessing/postprocessing;
- specific production task.

Before production deployment, it is therefore recommended to test the system using your own data and target hardware.

26. Intellectual Property

TAPe+ML is proprietary technology developed by **Comexp**.

The internal inference implementation and architecture source code are proprietary and are not disclosed to users. The product provides access to TAPe+ML functionality through the user interface and the APIs provided by the platform.

27. Support

TAPe+ML support covers questions related to:

- using the platform;
- Dataset preparation;
- Annotation;
- Auto-annotation;
- Fine-tuning;
- Validation;
- Detection;
- Segmentation;
- Classification;
- REST API;
- deployment.

For support, use the Comexp contact information provided on the TAPe+ML website.

Enterprise customers can receive extended technical support and assistance with deployment and integration.

28. Quick Start

If you are using TAPe+ML for the first time, start with a base model:

1. Open TAPe+ML.
2. Select **Detection, Segmentation, or Classification**.
3. Upload an image or connect a video stream via **SRT/RTSP**.
4. Check the inference result.
5. Create a **Custom Model** if necessary.
6. Integrate the result through **REST API or SSE**.

For your first evaluation, we recommend testing the base model on your own data.

If the base model does not solve your task, proceed to:

Custom Model → Annotation → Fine-tuning → Validation → Deployment