

dnt API Manual

None

None

None

Table of Contents

1. dnt API Manual	3
2. Quickstart	4
3. API Reference	5
3.1 API Reference	5
3.2 dnt Package	6
3.3 Detection	7
3.4 Tracking	0
3.5 Engine	0
3.6 Filter	0
3.7 Labeler	0
3.8 Shared Utilities	0

1. dnt API Manual

This site is generated with MkDocs and `mkdocstrings` from the Python source in `src/dnt`.

Use the navigation to browse package modules and APIs.

2. Quickstart

1. Install the package in your environment.
2. Run one of the examples in `examples/`.
3. Open the API reference for detailed class and function docs.

3. API Reference

3.1 API Reference

The pages in this section are rendered from module docstrings and object signatures.

3.2 dnt Package

DNT package for detection and tracking.

This package provides detection and tracking functionality.

3.3 Detection

3.3.1 Detection

Detection module for object detection using YOLO.

Detector

```
Detector(
    model: DetectorModel = DetectorModel.YOL026x,
    weights: str | None = None,
    conf: float = 0.25,
    nms: float = 0.7,
    max_det: int = 300,
    device: str = "auto",
    half: bool = False,
)
```

A wrapper around Ultralytics detection models for running object detection on videos and selected frames.

This class loads a YOLO (v8, v11, 26) or RT-DETR model from a local `models/` directory (or from a user-supplied .pt file) and provides convenience methods to:

- detect objects frame by frame in a video and return results as a pandas DataFrame,
- run detection only on specified frame indices,
- process a batch of videos and save per-video detection text files, and
- query basic video properties (FPS, frame count).

The detector automatically chooses an inference device (`cuda`, `xpu`, `mps`, or `cpu`) when `device="auto"`, and it can optionally enable half-precision inference on GPU.

Parameters:

Name	Type	Description	Default
<code>model</code>	<code>DetectorModel</code>	Built-in model weights to use (for example <code>DetectorModel.YOL026x</code>). Default is <code>DetectorModel.YOL026x</code> .	<code>YOL026x</code>
<code>weights</code>	<code>str</code>	Optional custom model weights to load. If relative, path is resolved under <code><module_dir>/models/</code> . Default is <code>None</code> .	<code>None</code>
<code>conf</code>	<code>float</code>	Confidence threshold for detections. Default is <code>0.25</code> .	<code>0.25</code>
<code>nms</code>	<code>float</code>	IoU / non-maximum suppression threshold. Default is <code>0.7</code> .	<code>0.7</code>
<code>max_det</code>	<code>int</code>	Maximum number of detections per frame. Default is <code>300</code> .	<code>300</code>
<code>device</code>	<code>(auto, cuda, xpu, cpu, mps)</code>	Inference device to use. If <code>"auto"</code> , the detector will pick an available accelerator first (<code>cuda → xpu → mps</code>) and fall back to CPU. Default is <code>"auto"</code> .	<code>"auto"</code>
<code>half</code>	<code>bool</code>	Whether to enable half-precision inference. This is only effective on GPU (CUDA). Default is <code>False</code> .	<code>False</code>

Notes ▾

- The class expects model weight files to be located under `<module_dir>/models/` when using the built-in weight names.
- Returned detection tables typically contain the columns: `frame`, `res`, `x`, `y`, `w`, `h`, `conf`, `class`.

Initialize a Detector for Ultralytics YOLO/RT-DETR models.

Parameters:

Name	Type	Description	Default
<code>model</code>	<code>DetectorModel</code>	Built-in model to use. Default is "yolo26x".	<code>YOLO26x</code>
<code>weights</code>	<code>str</code>	Customized model weights to load. Default is None, which means using the built-in weights in <code>model</code> choice.	<code>None</code>
<code>conf</code>	<code>float</code>	Confidence threshold. Default is 0.25.	0.25
<code>nms</code>	<code>float</code>	IoU/NMS threshold. Default is 0.7.	0.7
<code>max_det</code>	<code>int</code>	Maximum detections per frame. Default is 300. In crowded scenes, you may want to increase this.	300
<code>device</code>	<code>(auto, cuda, xpu, cp, u, mps)</code>	Inference device. Default is "auto".	<code>"auto"</code>
<code>half</code>	<code>bool</code>	Whether to use half precision (GPU only). Default is False.	<code>False</code>

Source code in `src/dnt/detect/yolo/detector.py` ▾

```

124     def __init__(  
125         self,  
126         model: DetectorModel = DetectorModel.YOLO26x,  
127         weights: str | None = None,  
128         conf: float = 0.25,  
129         nms: float = 0.7,  
130         max_det: int = 300,  
131         device: str = "auto",  
132         half: bool = False,  
133     ):  
134         """Initialize a Detector for Ultralytics YOLO/RT-DETR models.  
135  
136         Parameters  
137         -----  
138         model : DetectorModel, optional  
139             Built-in model to use. Default is "yolo26x".  
140         weights : str, optional  
141             Customized model weights to load.  
142             Default is None, which means using the built-in weights in `model` choice.  
143         conf : float, optional  
144             Confidence threshold. Default is 0.25.  
145         nms : float, optional  
146             IoU/NMS threshold. Default is 0.7.  
147         max_det : int, optional  
148             Maximum detections per frame.  
149             Default is 300. In crowded scenes, you may want to increase this.  
150         device : {"auto", "cuda", "xpu", "cpu", "mps"}, optional  
151             Inference device. Default is "auto".  
152         half : bool, optional  
153             Whether to use half precision (GPU only). Default is False.  
154  
155         """  
156         # Load model  
157         cwd = Path(__file__).parent.absolute()  
158         model_dir = cwd / "models"  
159         if not model_dir.exists():  
160             os.makedirs(model_dir)  
161  
162         if weights:  
163             model_path = Path(weights) if os.path.isabs(weights) else model_dir / weights  
164         else:  
165             model_path = model_dir / f"{model.value}"  
166  
167         # actually load model  
168         if ("yolo" in str(weights).lower()) or (model in YOLO_MODELS):  
169             self.model = YOLO(str(model_path))  
170         elif ("rtdetr" in str(weights).lower()) or (model in RTDETR_MODELS):  
171             self.model = RTDETR(str(model_path))  
172         else:  
173             raise ValueError(  
174                 f"Cannot infer model family from model={model} and weights={weights!r}. "  
175                 "Use a known DetectorModel or provide weights containing 'yolo' or 'rtdetr'. "  
176             )  
177         self.conf = conf  
178         self.nms = nms  
179         self.max_det = max_det  
180  
181         # device selection  
182         requested_device = str(device).lower().strip()  
183         requested_backend = requested_device.split(":", maxsplit=1)[0]  
184         valid_devices = {"auto", "cuda", "xpu", "mps", "cpu"}  
185         if requested_backend not in valid_devices:  
186             raise ValueError(  
187                 f"Invalid device={device!r}. Choose one of {sorted(valid_devices)} or backend:index like 'cuda:0'. "  
188             )  
189  
190         backend_available = {  
191             "cuda": torch.cuda.is_available(),  
192             "xpu": hasattr(torch, "xpu") and hasattr(torch.xpu, "is_available") and torch.xpu.is_available(),  
193             "mps": hasattr(torch.backends, "mps") and torch.backends.mps.is_available(),  
194             "cpu": True,  
195         }  
196  
197         if requested_backend == "auto":  
198             auto_priority = ("cuda", "xpu", "mps", "cpu")  
199             self.device = next(d for d in auto_priority if backend_available[d])  
200         else:  
201             self.device = requested_device if backend_available[requested_backend] else "cpu"  
202  
203         # half precision only makes sense on GPU  
204         self.half = half and (self.device == "cuda")

```

detect

```

detect(  
    input_video: str,  
    iou_file: str | None = None,  
    video_index: int | None = None,  
    video_tot: int | None = None,  
    start_frame: int | None = None,  
    end_frame: int | None = None,
)

```

```

verbose: bool = True,
show: bool = False,
message: str | None = None,
) -> pd.DataFrame

```

Run object detection on a video and return per-frame detections.

Parameters:

Name	Type	Description	Default
input_video	str	Path to the input video file.	required
iou_file	str	If provided, detection results are written to this file (CSV without header).	None
video_index	int	Index of this video in a batch, used only for progress display.	None
video_tot	int	Total number of videos in the batch, used only for progress display.	None
start_frame	int	Frame index to start detection from. If None or out of range, starts at 0.	None
end_frame	int	Frame index to stop detection at. If None or out of range, uses the last frame.	None
verbose	bool	Whether to show a progress bar. Default is True.	True
show	bool	Whether to display the video frames with detections. Default is False.	False
message	str None	Optional message shown in the progress bar description. Default is None.	None

Returns:

Type	Description
DataFrame	DataFrame with columns: <code>frame</code> , <code>res</code> , <code>x</code> , <code>y</code> , <code>w</code> , <code>h</code> , <code>conf</code> , <code>class</code> . If the video cannot be opened or no detections are found, an empty DataFrame with those columns is returned.

[Source code in `src/dnt/detect/yolo/detector.py`](#) ▾

206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305