Rigr AI Facial Age Estimation API — v3

Overview

The Rigr AI Facial Age Estimation API accepts images and returns estimated ages for all detected faces, along with bounding boxes, detection confidence scores, and calibrated uncertainty estimates. The current model (v3) achieves a Mean Absolute Error (MAE) of ~1.6 years on our benchmark dataset.

The API is available as a hosted cloud service or as a self-hosted Docker container for on-premise deployments (which can be air-gapped if required). Contact the Rigr.AI team for access and onboarding.

Cloud API Quick Start

The hosted cloud API is available at:

```
https://api.age.rigr.ai
```

To get started, request an API key from **age@email.rigr.ai**, then try:

```
# Health check (no auth required)
curl https://api.age.rigr.ai/healthz

# Verify your API key and check server info
curl -H "X-API-KEY: YOUR_KEY" https://api.age.rigr.ai/api/info

# Estimate age from an image
IMG_B64=$(base64 -w0 photo.jpg)
curl -X POST https://api.age.rigr.ai/api/image \
  -H "Content-Type: application/json" \
  -H "X-API-KEY: YOUR_KEY" \
  -d "{\"images\": [\"${IMG_B64}\"]}"
```

Or with the included Python client (see `examples/python_client.py`):

```
pip install httpx
python python_client.py --api-key YOUR_KEY --images photo.jpg
```

> **Note:** The cloud API may have a cold-start delay of a few seconds on the first request if the container has scaled to zero. Subsequent requests are fast.

For self-hosted / on-premise deployments, replace `https://api.age.rigr.ai` with your server's address (e.g., `http://localhost:8000`). See the Deployment section below.

Authentication

All API endpoints (except `/healthz`) require an API key provided via the `X-API-KEY` HTTP header.

```
X-API-KEY: your_api_key_here
```

API keys are issued by Rigr.AI. Contact us at **age@email.rigr.ai** to request one.

Endpoints

`GET /healthz`

Lightweight health check that verifies the edge/server is reachable. Does **not** require authentication and does not wake the container on the cloud deployment. Useful for uptime monitoring.

Response

Returns HTTP 200 with a plain text or JSON body indicating the service is healthy.

`POST /api/image`

Submit one or more images for age estimation. Images must be base64-encoded.

Request

Field	Type	Required	Description
`images`	`list[string]`	Yes	Base64-encoded images. At least 1 image is required.
`detection_threshold`	`float`	No	Face detection confidence threshold (0 to 1). Overrides the server default.
`max_dets_per_image`	`int`	No	Maximum number of face detections to return per image. Overrides the server default.

Example request body:

```
{
  "images": [
    "/9j/4AAQSk... (base64-encoded image)"
  ],
  "detection_threshold": 0.9,
  "max_dets_per_image": 10
}
```

Response

On success, the API returns HTTP 200 with a JSON body containing results for each submitted image, in the same order as the input.

Field	Type	Description
`error`	`object | null`	Top-level error, if a fatal error prevented processing.
`results`	`list[ImageResult]`	One entry per input image, in order.

Each **ImageResult** contains:

Field	Type	Description
`idx`	`int`	Index of this image in the input list.
`error`	`object | null`	

Field	Type	Description
		Per-image error (e.g., image could not be decoded). May be non-null even when results are present (e.g., truncated images are still processed).
`results`	`list[FaceResult]`	Detected faces in this image.

Each **FaceResult** contains:

Field	Type	Description
`idx`	`int`	Index of this face within the image.
`age`	`float`	Estimated age in years.
`uncertainty`	`float`	Calibrated uncertainty of the age estimate (in years). Higher values indicate lower model confidence.
`bbox`	`list[int]`	Bounding box of the detected face as `[x0, y0, x1, y1]` in pixels.
`score`	`float`	Face detection confidence score (0 to 1).
`source`	`string`	Source filename, if available.

Example successful response:

```
{
  "error": null,
  "results": [
    {
      "idx": 0,
      "error": null,
      "results": [
        {
          "idx": 0,
          "age": 25.3,
          "uncertainty": 1.2,
          "bbox": [175, 133, 364, 378],
          "score": 0.9998,
          "source": ""
        },
        {
          "idx": 1,
          "age": 10.5,
          "uncertainty": 2.1,
          "bbox": [633, 234, 725, 347],
```

```
          "score": 0.9993,
          "source": ""
        }
      ]
    }
  ]
}
```

`GET /api/info`

Returns information about the current pipeline configuration, including the model version and image constraints. Requires authentication via `X-API-KEY`.

Response

Field	Type	Description
`version`	`string`	Model version (e.g., `"v3"`).
`max_batch_size`	`int`	Maximum number of images per request.
`max_img_mb`	`float`	Maximum image file size in megabytes.
`min_img_px`	`int`	Minimum image dimension in pixels.
`max_img_px`	`int`	Maximum image dimension in pixels.
`detector_model`	`string`	Face detection model in use.
`detector_device`	`string`	Device running the face detector (e.g., `"cpu"`, `"cuda"`).
`estimator_device`	`string`	Device running the age estimator.
`estimator_model`	`string`	Age estimation model in use.

Example response:

```
{
  "version": "v3",
  "max_batch_size": 20,
  "max_img_mb": 50.0,
  "min_img_px": 24,
  "max_img_px": 6000,
  "detector_model": "retinaface_resnet50",
  "detector_device": "cuda",
  "estimator_device": "cuda",
```

```
  "estimator_model": "rigr_age_v3"
}
```

`GET /api/stats`

Returns throughput statistics for the age estimation pipeline. Requires authentication via `X-API-KEY`.

Response

Field	Type	Description
`detector_throughput`	`ThroughputStats`	Throughput statistics for the face detection stage.
`estimator_throughput`	`ThroughputStats`	Throughput statistics for the age estimation stage.
`total_throughput`	`ThroughputStats`	Combined end-to-end throughput statistics.

Each **ThroughputStats** object contains:

Field	Type	Description
`avg_task_time_sec`	`float`	Average time per task in seconds.
`items_per_sec`	`float`	Processing throughput in items per second.
`total_items`	`int`	Total number of items processed since server start.
`total_time_sec`	`float`	Total processing time in seconds.
`num_tasks`	`int`	Total number of tasks completed.

Example response:

```
{
  "detector_throughput": {
    "avg_task_time_sec": 0.045,
    "items_per_sec": 22.1,
    "total_items": 1500,
    "total_time_sec": 67.8,
    "num_tasks": 150
  },
  "estimator_throughput": {
    "avg_task_time_sec": 0.032,
```

```
    "items_per_sec": 31.2,
    "total_items": 1500,
    "total_time_sec": 48.1,
    "num_tasks": 150
  },
  "total_throughput": {
    "avg_task_time_sec": 0.077,
    "items_per_sec": 13.0,
    "total_items": 1500,
    "total_time_sec": 115.9,
    "num_tasks": 150
  }
}
```

Image Requirements

Constraint	Value
Supported formats	PNG, JPEG, JPG, BMP, GIF, WebP
Encoding	Base64
Maximum file size	50 MB per image
Maximum resolution	6000 x 6000 px
Minimum resolution	24 x 24 px

Error Handling

Errors are returned as structured JSON objects with `type`, `message`, and `detail` fields. Errors can occur at two levels:

- **Top-level** (`APIResponse.error`): Fatal errors that prevented the entire request from being processed.
- **Per-image** (`ImageResult.error`): Errors specific to a single image. The remaining images are still processed normally.

A per-image error may be present alongside results. For example, a truncated image will still be processed, but the `error` field will indicate the issue.

Error Codes

Code	HTTP Status	Description
`base64_decoding`	400	The provided string is not valid base64.
`image_loading`	400	The image could not be loaded or decoded.
`image_is_truncated`	400	The image is truncated (missing pixel data). The image is still processed, but results may be affected.
`batch_size_exceeded`	413	The number of images exceeds the maximum batch size.
`empty_images`	422	No images were provided in the request.
`image_size_error`	422	The image file size exceeds the maximum limit.
`image_too_large`	422	The image resolution exceeds the maximum allowed dimensions.
`image_too_small`	422	The image resolution is below the minimum required dimensions.
`model_runtime`	500	An error occurred during model inference.
`model_loading`	500	The model could not be loaded.

Authentication Errors

Code	HTTP Status	Description
`invalid_api_key`	401	The provided API key is invalid or expired.
`missing_api_key`	401	No `X-API-KEY` header was provided.

Validation Errors

If the request body does not conform to the expected schema, the API returns HTTP 422 with a structured error:

```
{
  "type": "RequestValidationError",
  "message": "Request validation failed",
  "detail": [
    {
```

```
      "loc": ["body", "images"],
      "message": "Missing field 'body.images'",
      "type": "missing"
    }
  ]
}
```

Example: Partial Failure

When some images fail but others succeed, results are returned for all images. Failed images have their `error` field populated:

```
{
  "error": null,
  "results": [
    {
      "idx": 0,
      "error": null,
      "results": [
        {
          "idx": 0,
          "age": 25.3,
          "uncertainty": 1.2,
          "bbox": [175, 133, 364, 378],
          "score": 0.9998,
          "source": ""
        }
      ]
    },
    {
      "idx": 1,
      "error": {
        "type": "image_too_small",
        "message": "Image too small -- minimum image size is 24px.",
        "detail": "Minimum image size is 24px, received 10x10px"
      },
      "results": []
    },
    {
      "idx": 2,
      "error": {
        "type": "image_is_truncated",
        "message": "Image appears to be truncated/is missing data.",
        "detail": "image file is truncated (4 bytes not processed)"
      },
      "results": [
        {
          "idx": 0,
          "age": 37.6,
```

```
          "uncertainty": 4.8,
          "bbox": [628, 239, 731, 348],
          "score": 0.9990,
          "source": ""
        }
      ]
    }
  ]
}
```

Deployment

Cloud (POC / evaluation)

The fastest way to evaluate the API is against the hosted cloud instance at `https://api.age.rigr.ai`. Request an API key from **age@email.rigr.ai** — no infrastructure setup is required. The cloud deployment runs the lite (ONNX-only, CPU) container on Cloudflare's edge network.

Self-hosted / on-premise

For production or air-gapped deployments, pre-built Docker images are available supporting both CPU and GPU (CUDA) inference. Contact the Rigr.AI team for registry access, image tags, and deployment guidance.

Image	Size	Use case
`rigrage-api:v3-cuda-12.9.1`	~8 GB	GPU inference (highest throughput)
`rigrage-api:v3-cpu`	~5 GB	CPU inference with PyTorch
`rigrage-api:v3-cpu-lite`	~1.6 GB	CPU inference, ONNX-only (smallest, fastest cold start)

All images expose the same API on port 8000. Replace the cloud URL with your server address:

```
curl -X POST http://localhost:8000/api/image \
  -H "Content-Type: application/json" \
  -H "X-API-KEY: YOUR_KEY" \
  -d '{"images": ["...base64..."]}'
```

Contact

For API access, support, or questions:

Age support: age@email.rigr.ai