Welcome to BFL Documentation!

Black Forest Labs builds state-of-the-art generative AI models

Our FLUX models offer state-of-the-art performance in media generation with top-of-the-line prompt following, visual quality, details and output diversity.

Our latest model, FLUX.1 Kontext, combines text-to-image generation with advanced image editing capabilities. Test it in our playground or read our editing guide.

What You Can Do

Transform text prompts into stunning images with exceptional quality and creativity using our **FLUX.1 Kontext, FLUX1.1 [pro], Ultra, and Raw models**. Transform images with **FLUX.1 Kontext** - our state-of-the-art editing model. Change colors, objects, add text, and more with text prompts Test our different models instantly in your browser without writing code Fine-tune FLUX Pro and FLUX Ultra on your specific style, objects, or concepts

Quick Start

Sign up to start using our services How to add credits to your account How to use our API to generate and edit images Master image editing with detailed examples and techniques

Ready to start creating? Try the Playground or make your first API call.

Image Generation with Text Prompts

> Complete guide to FLUX API endpoints for AI image generation. Learn text-to-image creation, API polling, regional endpoints, and code examples.

Our API endpoints enable media creation with BFL models. It follows an asynchronous design, where you first make a request for a generation and then query for the result of your request.

API Endpoints

Primary Global Endpoint

`api.bfl.ai` - Recommended for most use cases

- Routes requests across all available clusters globally
- Automatic failover between clusters for enhanced uptime
- Intelligent load distribution prevents bottlenecks during high traffic
- **Does not support finetuning and finetuned inference**
- **Important:** Always use the `polling_url` returned in responses when using this endpoint

Regional Endpoints

`api.eu.bfl.ai` - European Multi-cluster

- Multi-cluster routing limited to EU regions
- GDPR compliant
- **Does not support finetuning and finetuned inference**

`api.us.bfl.ai` - US Multi-cluster

- Multi-cluster routing limited to US regions
- **Does not support finetuning and finetuned inference**

Legacy Regional Endpoints

`api.eu1.bfl.ai` - EU Single-cluster

- Single cluster, no automatic failover
- Required for finetuning and finetuned inference operations in EU region

`api.us1.bfl.ai` - US Single-cluster

- Single cluster, no automatic failover
- Required for finetuning and finetuned inference operations in US region

For enhanced reliability and performance, we recommend using the global endpoint `api.bfl.ai` or regional endpoints `api.eu.bfl.ai`/`api.us.bfl.ai` for inference tasks.

Available Endpoints

We currently support the following endpoints for image generation:

1. `/flux-kontext-pro`
2. `/flux-kontext-max`
3. `/flux-pro-1.1-ultra`
4. `/flux-pro-1.1`
5. `/flux-pro`
6. `/flux-dev`

Create Your First Image

Submit Generation Request

To submit an image generation task with FLUX 1.1 [pro], create a request:

```
bash submit_request.sh # Install curl and jq, then run: # Make sure to
set your API key: export BFL_API_KEY="your_key_here" request=$(curl -X
'POST' \ 'https://api.bfl.ai/v1/flux-kontext-pro' \ -H 'accept:
application/json' \ -H "x-key: ${BFL_API_KEY}" \ -H 'Content-Type:
application/json' \ -d '{ "prompt": "A cat on its back legs running
like a human is holding a big silver fish with its arms. The cat is
running away from the shop owner and has a panicked look on his face.
The scene is situated in a crowded market.", "aspect_ratio": "1:1",
}') echo $request request_id=$(jq -r .id <<< $request)
polling_url=$(jq -r .polling_url <<< $request) echo "Request ID:
${request_id}" echo "Polling URL: ${polling_url}" python
submit_request.py # Install requests: pip install requests import os
import requests request = requests.post(
'https://api.bfl.ai/v1/flux-kontext-pro', headers={ 'accept':
'application/json', 'x-key': os.environ.get("BFL_API_KEY"),
'Content-Type': 'application/json', }, json={ 'prompt': 'A cat on its
back legs running like a human is holding a big silver fish with its
arms. The cat is running away from the shop owner and has a panicked
look on his face. The scene is situated in a crowded market.',
"aspect_ratio": "1:1" }, ).json() print(request) request_id =
request["id"] polling_url = request["polling_url"] print(f"Request ID:
{request_id}") print(f"Polling URL: {polling_url}")
```

A successful response will be a json object containing the request's id and a `polling_url` that should be used to retrieve the result.

Important: When using the global endpoint (`api.bfl.ai`) or regional endpoints (`api.eu.bfl.ai`, `api.us.bfl.ai`), you must use the `polling_url` returned in the response for checking request status.

Poll for Results

To retrieve the result, poll the endpoint using the `polling_url`:

```
bash poll_results.sh # This assumes that the request_id and
polling_url variables are set from the previous step while true do
sleep 0.5 result=$(curl -s -X 'GET' \ "${polling_url}" \ -H 'accept:
application/json' \ -H "x-key: ${BFL_API_KEY}") status=$(jq -r .status
<<< $result) echo "Status: $status" if [ "$status" == "Ready" ] then
echo "Result: $(jq -r .result.sample <<< $result)" break elif [
"$status" == "Error" ] || [ "$status" == "Failed" ] then echo
"Generation failed: $result" break fi done python poll_results.py #
This assumes request_id and polling_url are set from the previous step
import time while True: time.sleep(0.5) result = requests.get(
polling_url, headers={ 'accept': 'application/json', 'x-key':
os.environ.get("BFL_API_KEY"), }, params={ 'id': request_id, },
).json() status = result["status"] print(f"Status: {status}") if
status == "Ready": print(f"Result: {result['result']['sample']}")
break elif status in ["Error", "Failed"]: print(f"Generation failed:
{result}") break
```

A successful response will be a JSON object containing the result, where `result['sample']` is a signed URL for retrieval.

Our signed URLs are only valid for 10 minutes. Please retrieve your result within this timeframe.
Image Delivery: The `result.sample` URLs are served from delivery endpoints

(`delivery-eu1.bfl.ai`, `delivery-us1.bfl.ai`) and are not meant to be served directly to users. We recommend downloading the image and re-serving it from your own infrastructure. We do not enable CORS on delivery URLs.

See our reference documentation for a full list of options and our inference repo.

Limits

Rate Limits: Sending requests to our API is limited to 24 active tasks. If you exceed your limit, you'll receive a status code `429` and must wait until one of your previous tasks has finished.
Rate Limits: Additionally, due to capacity issues, for `flux-kontext-max`, the requests to our API is limited to 6 active tasks. **Credits:** If you run out of credits (status code `402`), visit https://api.bfl.ai, sign in and click "Add" to buy additional credits. See also managing your account. If you require higher volumes, please contact us at flux@blackforestlabs.ai

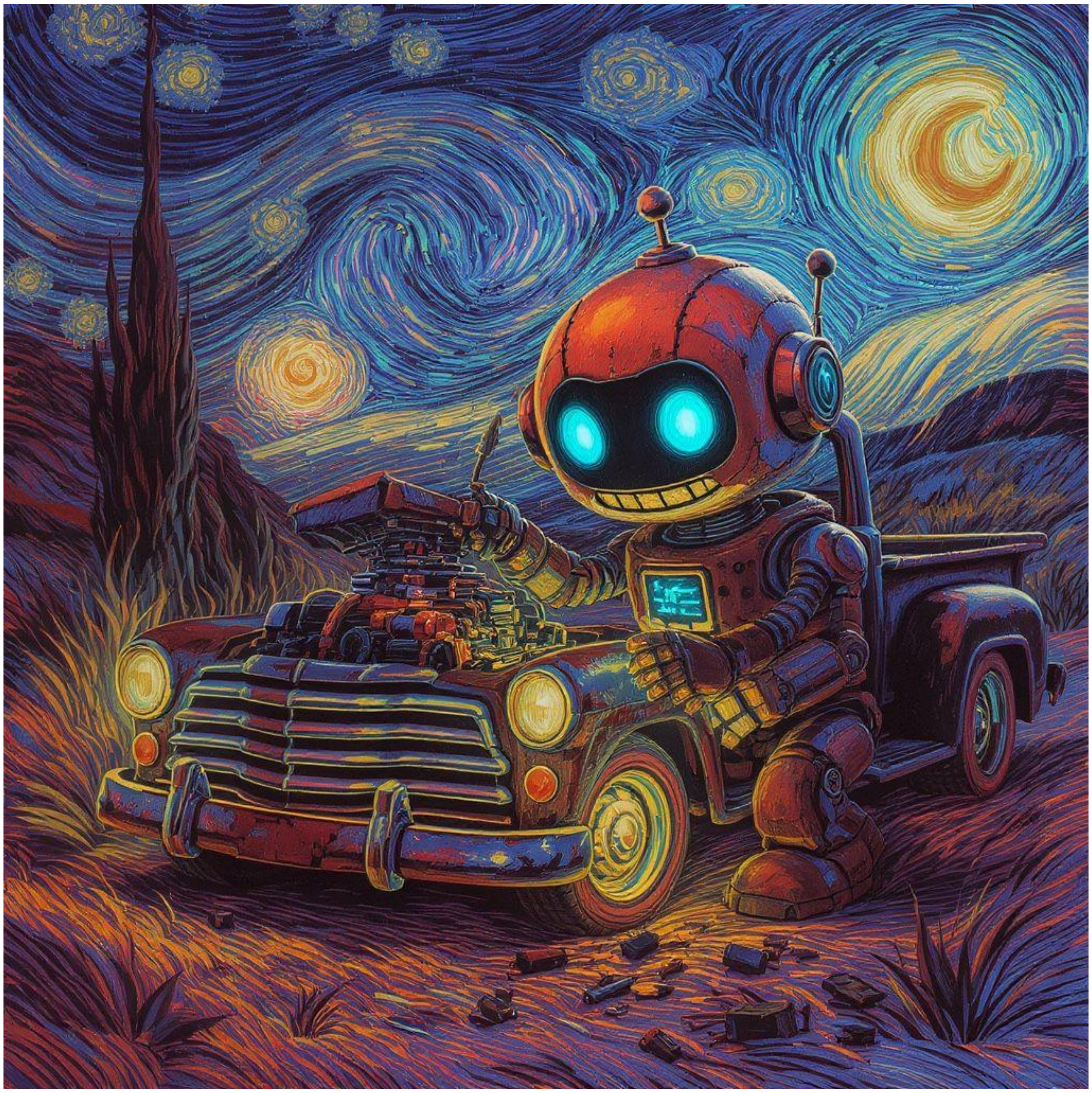
Image Generation

FLUX.1 Kontext [pro] can generate images directly from text input, allowing you to create entirely new visuals. This guide focuses on using the `/flux-kontext-pro` endpoint for its Text-to-Image capabilities.

To generate an image from text, you'll make a request to the `/flux-kontext-pro` endpoint.

Examples of Image Generation

FLUX.1 Kontext [pro] not only can edit images, but it can also generate images with a prompt. Here are a few examples of what you can create:



Prompts for the images above:

• **Abstract cat artwork:** "Abstract expressionist painting Pop Art and cubism early 20 century, straight lines and solids, cute cat face without body, warm colors, green, intricate details, hologram floating in space, a vibrant digital illustration, black background, flat color, 2D, strong lines."

• **Robot and truck:** "A cute round rusted robot repairing a classic pickup truck, colorful, futuristic, vibrant glow, van gogh style"

• **Furry elephant:** "A small furry elephant pet looks out from a cat house"

- **Face paint portrait:** "A close-up of a face adorned with intricate black and blue patterns. The left side of the face is predominantly yellow, with symbols and doodles, while the right side is dark, featuring mechanical elements. The eye on the left is a striking shade of yellow, contrasting sharply with the surrounding patterns. The face is partially covered by a hooded garment, realistic style"

FLUX

Prompts for the images above:

• **Rainy car scene:** "Close-up of a vintage car hood under heavy rain, droplets cascading down the deep cherry-red paint, windshield blurred with streaks of water, glowing headlights diffused through mist, reflections of crimson neon signage spelling "FLUX" dancing across the wet chrome grille, steam rising from the engine, ambient red light enveloping the scene, moody composition, shallow depth of field, monochromatic red palette, cinematic lighting with glossy textures."

• **Burning temple warrior:** "A lone warrior, clad in bloodstained samurai armor, stands motionless before a massive pagoda engulfed in flames. Embers and ash swirl around him like ghosts of fallen enemies. The once-sacred temple is collapsing, its ornate carvings crumbling into the blaze as distant screams echo through the smoke-filled air. A tattered banner flutters beside him, the last symbol of a forgotten oath. The scene is both devastating and mesmerizing, with deep reds, burning oranges, and cold blue shadows creating a stark contrast. Cinematic composition, ultra-detailed textures, dynamic lighting, atmospheric fog, embers in the wind, dark fantasy realism, intense contrast."

• **Foggy gas station:** "Remote gas station swallowed by crimson fog, green glow from overhead lights staining the asphalt, new tiny smart car idling with taillights cutting through the mist, vending machine humming beside cracked fuel pumps, oily puddles reflecting distorted neon, shadows stretching unnaturally long, skeletal trees barely visible in the background, wide-angle cinematic shot, deep green monochromatic palette with faint charcoal accents, backlighting and heavy atmosphere, surreal and ominous mood."

• **Detective game character:** "Retro game style, man in old school suit, upper body, true detective, detailed character, nigh sky, crimson moon silhouette, american muscle car parked on dark street in background, complex background in style of Bill Sienkiewicz and Dave McKean and Carne Griffiths, extremely detailed, mysterious, grim, provocative, thrilling, dynamic, action-packed, fallout style, vintage, game theme, masterpiece, high contrast, stark. vivid colors, 16-bit, pixelated, textured, distressed"

Using FLUX.1 Kontext API for Text-to-Image Generation

Create a Request

```
bash create_request.sh # Install `curl` and `jq`, then run: # Ensure
BFL_API_KEY is set # export BFL_API_KEY="your_api_key_here"
request=$(curl -X POST \ 'https://api.bfl.ai/flux-kontext-pro' \ -H
'accept: application/json' \ -H "x-key: ${BFL_API_KEY}" \ -H
'Content-Type: application/json' \ -d '{ "prompt": "A small furry
elephant pet looks out from a cat house", "width": 1024, "height":
1024 }') echo "Full request response:" echo $request request_id=$(jq
-r .id <<< $request) echo "Request ID: ${request_id}" python
create_request.py # Install `requests` (e.g. `pip install requests`)
and `Pillow` (e.g. `pip install Pillow`), then run: import os import
requests import base64 from PIL import Image from io import BytesIO
request = requests.post( 'https://api.bfl.ai/v1/flux-kontext-pro',
headers={ 'accept': 'application/json', 'x-key':
os.environ.get("BFL_API_KEY"), 'Content-Type': 'application/json', },
json={ 'prompt': 'A small furry elephant pet looks out from a cat
house', }, ).json() print(request) request_id = request["id"]
```

A successful response will be a JSON object containing the request's `id`. This ID is used to retrieve the generated image.

Poll for Result

After submitting a request, you need to poll the `/v1/get_result` endpoint using the `request_id` to check the status and retrieve the output when ready.

```
bash poll_result.sh while true do sleep 1.5 result=$(curl -s -X 'GET'
\ "https://api.bfl.ai/v1/get_result?id=${request_id}" \ -H 'accept:
application/json' \ -H "x-key: ${BFL_API_KEY}") status=$(jq -r .status
<<< $result) echo "Status: $status" if [ "$status" == "Ready" ] then
echo "Result: $(jq -r .result.sample <<< $result)" break elif [
"$status" != "Processing" ] && [ "$status" != "Queued" ] then echo "An
error or unexpected status occurred: $result" break fi done python
poll_result.py # This assumes that the `request_id` variable is set.
import time import os import requests # Ensure request_id is set from
the previous step # request_id = "your_request_id_here" while True:
time.sleep(1.5) result = requests.get(
'https://api.bfl.ai/v1/get_result', headers={ 'accept':
'application/json', 'x-key': os.environ.get("BFL_API_KEY"), },
params={'id': request_id}, ).json() status = result.get("status")
print(f"Status: {status}") if status == "Ready": print(f"Result:
{result.get('result', {}).get('sample')}") break elif status not in
["Processing", "Queued"]: print(f"An error or unexpected status
occurred: {result}") break
```

A successful response will be a json object containing the result, and `result['sample']` is a signed URL for retrieval.

Our signed URLs are only valid for 10 minutes. Please retrieve your result within this timeframe.

FLUX.1 Kontext Text-to-Image Parameters

FLUX.1 Kontext creates 1024x1024 images by default. Use `aspect_ratio` to adjust the dimensions while keeping the same total pixels.

- **Supported Range**: Aspect ratios can range from 3:7 (portrait) to 7:3 (landscape).
- **Default Behavior**: If `aspect_ratio` is not specified, the model will default to a standard aspect ratio like 1:1 (e.g. 1024x1024).

Parameter	Type	Default	Description	Required
`prompt`	string		Text description of the desired image.	**Yes**
`aspect_ratio`	string / null	`"1:1"`	Desired aspect ratio (e.g., "16:9"). All outputs are ~1MP total. Supports ratios from 3:7 to 7:3.	No
`seed`	integer / null	`null`	Seed for reproducibility. If `null` or omitted, a random seed is used. Accepts any integer.	No
`prompt_upsampling`	boolean	`false`	If true, performs upsampling on the prompt	No
`safety_tolerance`	integer	`2`	Moderation level for inputs and outputs. Value ranges from 0 (most strict) to 6 (more permissive).	No
`output_format`	string	`"jpeg"`	Desired format of the output image. Can be "jpeg" or "png".	No
`webhook_url`	string / null	`null`	URL for asynchronous completion notification. Must be a valid HTTP/HTTPS URL.	No
`webhook_secret`	string / null	`null`	Secret for webhook signature verification, sent in the `X-Webhook-Secret` header.	No

Image Editing

FLUX.1 Kontext [pro] is a model designed for Text-to-Image generation and **advanced Image Editing**. This guide focuses on its Image Editing capabilities. Unlike other models, you don't need to fine-tune or create complex workflows to achieve this - Flux.1 Kontext [pro] handles it out of the box.

Kontext's image editing, accessed via the `/flux-kontext-pro` endpoint, provides the following key functionalities:

- **Simple Editing**: Change specific parts of an image while keeping the rest untouched
- **Smart Changes**: Make edits that look natural and fit with the rest of the image

- **Text in Images**: Add or modify text within your images

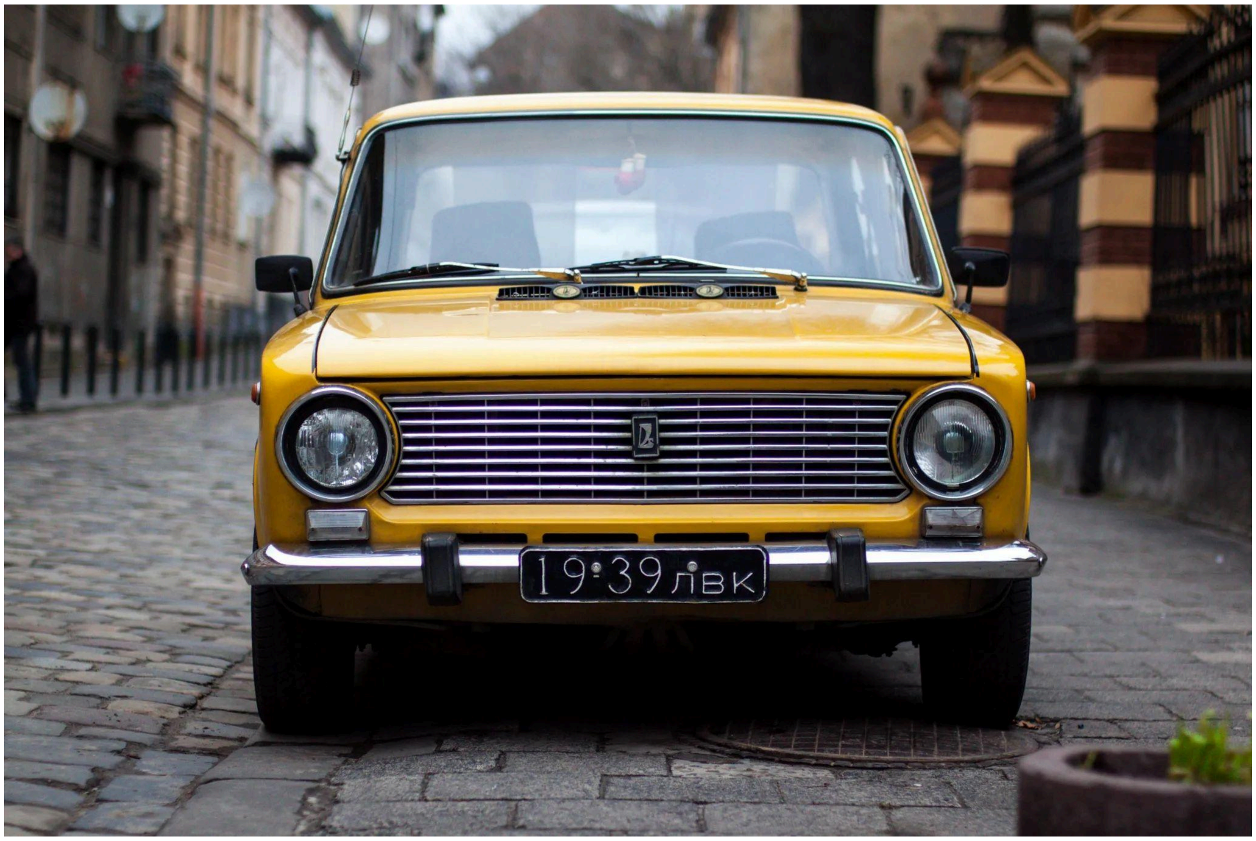
For comprehensive prompting techniques and advanced editing strategies, see our detailed Prompting Guide - Image-to-Image.

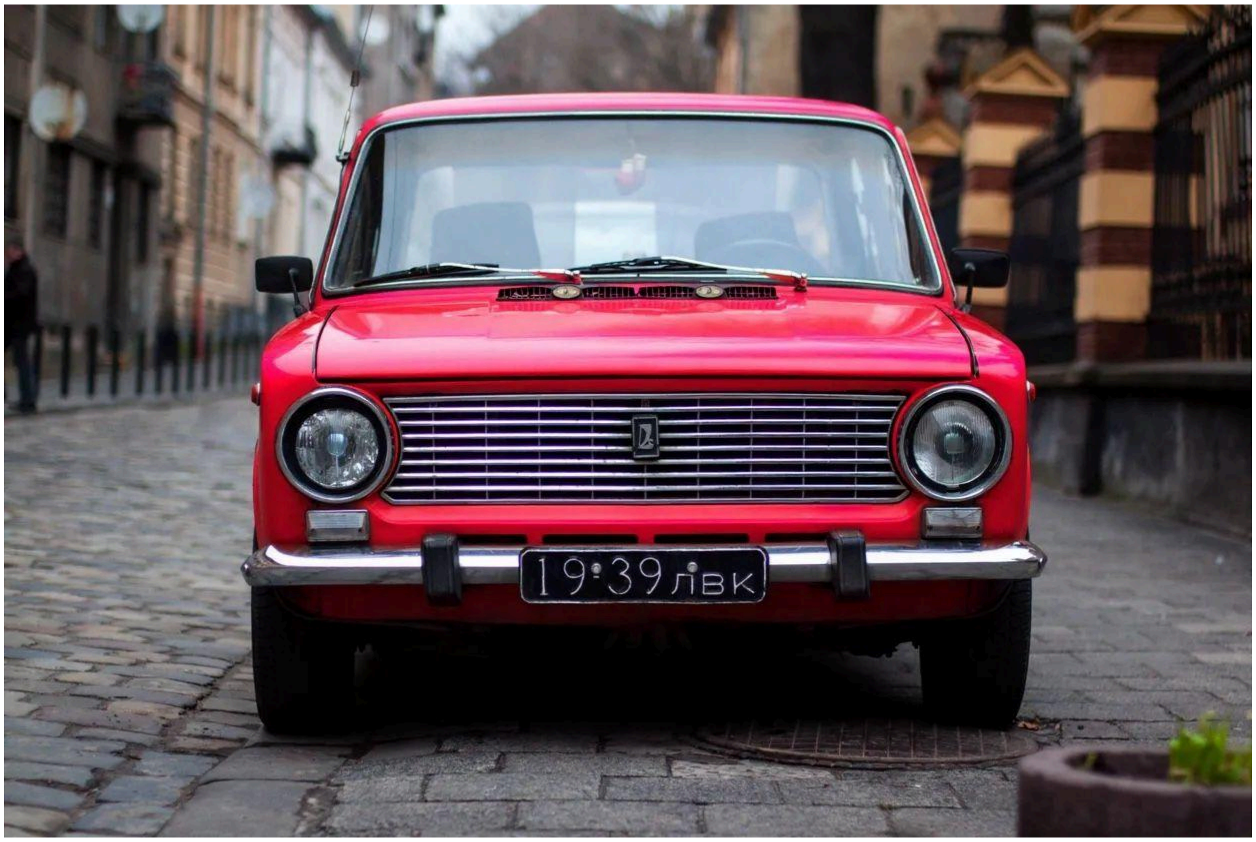
Examples of Editing

Basic Object Modifications

FLUX.1 Kontext is really good at straightforward object modification, for example if we want to change the colour of an object, we can prompt it.

For example: `Change the car color to red`


19·39лвк


19·39лвк

Iterative Editing

FLUX.1 Kontex excels at character consistency, even after multiple edits. Starting from a reference picture, we can see that the character is consistent throughout the sequence.

Text Editing

FLUX.1 Kontext can directly edit text that appears in images, making it easy to update signs, posters, labels, and more without recreating the entire image.

The most effective way to edit text is using quotation marks around the specific text you want to change:

Prompt Structure: `Replace '[original text]' with '[new text]'`

Example - We can see below where we have an input image with "Choose joy" written, and we replace "joy" with "BFL" - note the upper case format for BFL.

Choose
joy

Choose
BFL

SYNC & BLOOM
SYNC & BLOOM

FLUX & JOY
FLUX & JOY

Using FLUX.1 Kontext API for Image Editing

This **requires both** a **text prompt** and **an input image** to work, with the input image serving as the base that will be edited according to your prompt.

To use Kontext for image editing, you'll make a request to the `/flux-kontext-pro` endpoint:

Create Request

```
bash create_request.sh # Install `curl` and `jq`, then run:
request=$(curl -X POST \ 'https://api.bfl.ai/v1/flux-kontext-pro' \ -H
'accept: application/json' \ -H "x-key: ${BFL_API_KEY}" \ -H
'Content-Type: application/json' \ -d '{ "prompt": "<What you want to
edit on the image>", "input_image": "<base64 converted image>", }')
echo $request request_id=$(jq -r .id <<< $request) python
create_request.py # Install `requests` (e.g. `pip install requests`) #
and `Pillow` (e.g. `pip install Pillow`) import os import requests
import base64 from PIL import Image from io import BytesIO # Load and
encode your image # Replace "<your_image.jpg>" with the path to your
image file image = Image.open("<your_image.jpg>") buffered = BytesIO()
image.save(buffered, format="JPEG") # Or "PNG" if your image is PNG
img_str = base64.b64encode(buffered.getvalue()).decode() request =
requests.post( 'https://api.bfl.ai/v1/flux-kontext-pro', headers={
'accept': 'application/json', 'x-key': os.environ.get("BFL_API_KEY"),
'Content-Type': 'application/json', }, json={ 'prompt': '<What you
want to edit on the image>', 'input_image': img_str, }, ).json()
print(request) request_id = request["id"]
```

A successful response will be a json object containing the request's id, that will be used to retrieve the actual result.

Poll for Result

After submitting a request, you need to poll the `/v1/get_result` endpoint using the `request_id` to check the status and retrieve the output when ready.

```
bash poll_result.sh while true do sleep 1.5 result=$(curl -s -X 'GET'
\ "https://api.bfl.ai/v1/get_result?id=${request_id}" \ -H 'accept:
application/json' \ -H "x-key: ${BFL_API_KEY}") status=$(jq -r .status
<<< $result) echo "Status: $status" if [ "$status" == "Ready" ] then
echo "Result: $(jq -r .result.sample <<< $result)" break elif [
"$status" != "Processing" ] && [ "$status" != "Queued" ] then echo "An
error or unexpected status occurred: $result" break fi done python
poll_result.py # This assumes that the `request_id` variable is set.
import time import os import requests # Ensure request_id is set from
the previous step # request_id = "your_request_id_here" while True:
time.sleep(1.5) result = requests.get(
'https://api.bfl.ai/v1/get_result', headers={ 'accept':
'application/json', 'x-key': os.environ.get("BFL_API_KEY"), },
params={'id': request_id}, ).json() status = result.get("status")
print(f"Status: {status}") if status == "Ready": print(f"Result:
{result.get('result', {}).get('sample')}") break elif status not in
["Processing", "Queued"]: print(f"An error or unexpected status
occurred: {result}") break
```

A successful response will be a json object containing the result, and `result['sample']` is a signed URL for retrieval.

Our signed URLs are only valid for 10 minutes. Please retrieve your result within this timeframe.

FLUX.1 Kontext Image Editing Parameters (for /flux-kontext-pro)

FLUX.1 Kontext creates 1024x1024 images by default. Use `aspect_ratio` to adjust the dimensions while keeping the same total pixels.

- **Supported Range**: Aspect ratios can range from 3:7 (portrait) to 7:3 (landscape).
- **Default Behavior**: If `aspect_ratio` is not specified, the model will default to a standard aspect ratio like 1:1 (e.g. 1024x1024).

List of Kontext parameters for image editing via the `/flux-kontext-pro` endpoint:

Parameter	Type	Default	Description	Required
`prompt`	string		Text description of the edit to be applied.	**Yes**
`input_image`	string		Base64 encoded image to use as reference. Supports up to 20MB or 20 megapixels.	**Yes**

`aspect_ratio`	string / null	`"1:1"`	Desired aspect ratio (e.g., "16:9"). All outputs are ~1MP total. Supports ratios from 3:7 to 7:3.	No
`seed`	integer / null	`null`	Seed for reproducibility. If `null` or omitted, a random seed is used. Accepts any integer.	No
`prompt_upsampling`	boolean	`false`	If true, performs upsampling on the prompt	No
`safety_tolerance`	integer	`2`	Moderation level for inputs and outputs. Value ranges from 0 (most strict) to 2 (balanced)	No
`output_format`	string	`"jpeg"`	Desired format of the output image. Can be "jpeg" or "png".	No
`webhook_url`	string / null	`null`	URL for asynchronous completion notification. Must be a valid HTTP/HTTPS URL.	No
`webhook_secret`	string / null	`null`	Secret for webhook signature verification, sent in the `X-Webhook-Secret` header.	No

FLUX Finetuning Guide

The BFL Finetuning API enables you to customize FLUX Pro and FLUX Ultra using 1-20 images of your own visual content, and optionally, text descriptions.

Regional Requirements for Finetuning

Important: Finetuning operations must use region-specific endpoints and cannot use the global or multi-cluster regional endpoints.

Finetuning Endpoints

EU Region: `api.eu1.bfl.ai`

- Use this endpoint for all finetuning operations in the EU region
- Required for GDPR compliance when training with EU data

US Region: `api.us1.bfl.ai`

- Use this endpoint for all finetuning operations in the US region

Regional Consistency

You can only run inference on finetuned models in the same region where they were trained. When you submit a finetuning task using `https://api.us1.bfl.ai/v1/finetune`, you must use `https://api.us1.bfl.ai/v1/flux-pro-1.1-ultra-finetuned` for inference. Cross-region inference will not find your finetuned model.

Getting Started: Step-by-Step Guide

Create a local folder containing your training images: ***Supported formats:*** *JPG, JPEG, PNG, and WebP* **Recommended:** More than 5 images High-quality datasets with clear, articulated subjects/objects/styles significantly improve training results. Higher resolution source images help but are capped at 1MP. Create text files with descriptions for your images: *Text files should share the same name as their corresponding images* **Example:** If your image is `sample.jpg`, create `sample.txt` Compress your folder into a ZIP file containing all images and optional text descriptions. Select appropriate hyperparameters based on your use case. See the Training Parameters section below for detailed configuration options. Use the provided Python functions to submit your finetuning task to the BFL API. Check the status of your training job using the progress monitoring functions. Once training is complete, use your custom model through the available finetuned endpoints.

Model Training Parameters

Required Parameters

Determines the finetuning approach based on your concept. **Options:** `"character"`, `"product"`, `"style"`, `"general"` In "general" mode, the entire image is captioned when captioning is True without specific focus areas. No subject specific improvements will be made. Descriptive note to identify your fine-tune since names are UUIDs. Will be displayed in `finetune_details`.

Optional Parameters

Minimum: 100 Defines training duration. For fast exploration, 100-150 iterations can be enough. For more complex concepts, larger datasets, or extreme precision, more iterations than the default can help. **Default:** 0.00001 if `finetune_type` is "full", 0.0001 if `finetune_type` is "lora" Lower values can improve the result but might need more iterations to learn a concept. Higher values can allow you to train for less iterations at a potential loss in quality. For `finetune_type` "lora", values 10 times larger than for "full" are recommended. **Options:** `"speed"`, `"quality"`, `"high_res_only"` The speed priority will improve speed per training step. Enables/disables automatic image captioning. Unique word/phrase that will be used in the captions to reference the newly introduced concepts. Choose between 32 and 16. A lora_rank of 16 can increase training efficiency and decrease loading times. Choose between "full" for a full finetuning + post hoc extraction of the trained weights into a LoRA or "lora" for a raw LoRA training.

Inference Endpoints

Available endpoints for your finetuned model:

1. `/flux-pro-1.1-ultra-finetuned`
2. `/flux-pro-finetuned`
3. `/flux-pro-1.0-depth-finetuned`
4. `/flux-pro-1.0-canny-finetuned`
5. `/flux-pro-1.0-fill-finetuned`

Regional Inference Requirements

Regional Consistency Required: You can only inference the finetunes in the region that you trained them in. ***EU Finetuned Models:*** *Submit finetuning using:* ***https://api.us1.bfl.ai/v1/finetune****, query using* ***https://api.us1.bfl.ai/v1/flux-pro-1.1-ultra-finetuned*** **US Finetuned Models:** Submit finetuning using: **https://api.eu1.bfl.ai/v1/finetune**, query using **https://api.eu1.bfl.ai/v1/flux-pro-1.1-ultra-finetuned** Querying the global endpoint `https://api.bfl.ai/v1/flux-pro-1.1-ultra-finetuned` will not find your region-specific finetune.

Additional Inference Parameters

The endpoints have additionally all input parameters that their non-finetuned sibling endpoints have. References your specific model. Find the `finetune_id` either in `my_finetunes` or in the return dict of your `/finetune` POST. **Range:** 0-2 Controls finetune influence. Increase this value if your target concept isn't showing up strongly enough. The optimal setting depends on your finetune and prompt.

Implementation Guide

Setup and Dependencies

`bash pip install requests fire` `bash export BFL_API_KEY="your_api_key_here"`

Implementation Guide

Example Python Implementation

```
python bfl_finetune.py Assuming you have prepared your images in a
`finetuning.zip` file: # submit finetuning task $ python
bfl_finetune.py request_finetuning finetuning.zip myfirstfinetune id:
<finetune_id> # query status $ python bfl_finetune.py
finetune_progress <finetune_id> id: <finetune_id> status: Pending
result: null progress: null # once status shows Ready, run inference
(defaults to flux-pro-1.1-ultra-finetuned) $ python bfl_finetune.py
finetune_inference <finetune_id> --prompt="image of a TOK"
finetune_id: <inference_id> # retrieve inference result $ python
bfl_finetune.py get_inference <inference_id> id: <inference_id>
status: Ready result: {"sample": <result_url>, "prompt": "image of a
TOK"} progress: null """ import os import base64 import requests def
request_finetuning( zip_path, finetune_comment, trigger_word="TOK",
mode="general", api_key=None, iterations=300, learning_rate=0.00001,
captioning=True, priority="quality", finetune_type="full",
lora_rank=32, ): """ Request a finetuning using the provided ZIP file.
Args: zip_path (str): Path to the ZIP file containing training data
finetune_comment (str): Comment for the finetune_details trigger_word
(str): Trigger word for the model mode (str): Mode for caption
generation api_key (str): API key for authentication iterations (int):
Number of training iterations learning_rate (float): Learning rate for
optimization captioning (bool): Enable/disable auto-captioning
priority (str): Training quality setting lora_rank (int): Lora rank
finetune_type (str): "full" or "lora" Returns: dict: API response
Raises: FileNotFoundError: If ZIP file is missing
requests.exceptions.RequestException: If API request fails """ if
api_key is None: if "BFL_API_KEY" not in os.environ: raise ValueError(
"Provide your API key via --api_key or an environment variable
BFL_API_KEY" ) api_key = os.environ["BFL_API_KEY"] if not
os.path.exists(zip_path): raise FileNotFoundError(f"ZIP file not found
at {zip_path}") assert mode in ["character", "product", "style",
"general"] with open(zip_path, "rb") as file: encoded_zip =
base64.b64encode(file.read()).decode("utf-8") url =
"https://api.us1.bfl.ai/v1/finetune" headers = { "Content-Type":
"application/json", "X-Key": api_key, } payload = {
"finetune_comment": finetune_comment, "trigger_word": trigger_word,
"file_data": encoded_zip, "iterations": iterations, "mode": mode,
"learning_rate": learning_rate, "captioning": captioning, "priority":
priority, "lora_rank": lora_rank, "finetune_type": finetune_type, }
response = requests.post(url, headers=headers, json=payload) try:
response.raise_for_status() return response.json() except
requests.exceptions.RequestException as e: raise
requests.exceptions.RequestException( f"Finetune request
failed:\n{str(e)}\n{response.content.decode()}" ) def
finetune_progress( finetune_id, api_key=None, ): if api_key is None:
if "BFL_API_KEY" not in os.environ: raise ValueError( "Provide your
API key via --api_key or an environment variable BFL_API_KEY" )
api_key = os.environ["BFL_API_KEY"] url =
"https://api.us1.bfl.ai/v1/get_result" headers = { "Content-Type":
"application/json", "X-Key": api_key, } payload = { "id": finetune_id,
} response = requests.get(url, headers=headers, params=payload) try:
response.raise_for_status() return response.json() except
requests.exceptions.RequestException as e: raise
requests.exceptions.RequestException( f"Finetune progress
failed:\n{str(e)}\n{response.content.decode()}" ) def finetune_list(
api_key=None, ): if api_key is None: if "BFL_API_KEY" not in
os.environ: raise ValueError( "Provide your API key via --api_key or
an environment variable BFL_API_KEY" ) api_key =
os.environ["BFL_API_KEY"] url =
"https://api.us1.bfl.ai/v1/my_finetunes" headers = { "Content-Type":
"application/json", "X-Key": api_key, } response = requests.get(url,
headers=headers) try: response.raise_for_status() return
response.json() except requests.exceptions.RequestException as e:
raise requests.exceptions.RequestException( f"Finetune listing
failed:\n{str(e)}\n{response.content.decode()}" ) def
finetune_details( finetune_id, api_key=None, ): if api_key is None: if
"BFL_API_KEY" not in os.environ: raise ValueError( "Provide your API
key via --api_key or an environment variable BFL_API_KEY" ) api_key =
os.environ["BFL_API_KEY"] url =
"https://api.us1.bfl.ai/v1/finetune_details" headers = {
"Content-Type": "application/json", "X-Key": api_key, } payload = {
"finetune_id": finetune_id, } response = requests.get(url,
headers=headers, params=payload) try: response.raise_for_status()
return response.json() except requests.exceptions.RequestException as
e: raise requests.exceptions.RequestException( f"Finetune details
failed:\n{str(e)}\n{response.content.decode()}" ) def finetune_delete(
finetune_id, api_key=None, ): if api_key is None: if "BFL_API_KEY" not
in os.environ: raise ValueError( "Provide your API key via --api_key
or an environment variable BFL_API_KEY" ) api_key =
os.environ["BFL_API_KEY"] url =
"https://api.us1.bfl.ai/v1/delete_finetune" headers = {
"Content-Type": "application/json", "X-Key": api_key, } payload = {
"finetune_id": finetune_id, } response = requests.post(url,
headers=headers, json=payload) try: response.raise_for_status() return
response.json() except requests.exceptions.RequestException as e:
raise requests.exceptions.RequestException( f"Finetune deletion
failed:\n{str(e)}\n{response.content.decode()}" ) def
finetune_inference( finetune_id, finetune_strength=1.2,
endpoint="flux-pro-1.1-ultra-finetuned", api_key=None, **kwargs, ): if
api_key is None: if "BFL_API_KEY" not in os.environ: raise ValueError(
"Provide your API key via --api_key or an environment variable
BFL_API_KEY" ) api_key = os.environ["BFL_API_KEY"] url =
f"https://api.us1.bfl.ai/v1/{endpoint}" headers = { "Content-Type":
"application/json", "X-Key": api_key, } payload = { "finetune_id":
finetune_id, "finetune_strength": finetune_strength, **kwargs, }
response = requests.post(url, headers=headers, json=payload) try:
response.raise_for_status() return response.json() except
requests.exceptions.RequestException as e: raise
requests.exceptions.RequestException( f"Finetune inference
failed:\n{str(e)}\n{response.content.decode()}" ) def get_inference(
id, api_key=None, ): if api_key is None: if "BFL_API_KEY" not in
os.environ: raise ValueError( "Provide your API key via --api_key or
an environment variable BFL_API_KEY" ) api_key =
os.environ["BFL_API_KEY"] url = "https://api.us1.bfl.ai/v1/get_result"
headers = { "Content-Type": "application/json", "X-Key": api_key, }
payload = { "id": id, } response = requests.get(url, headers=headers,
params=payload) try: response.raise_for_status() return
response.json() except requests.exceptions.RequestException as e:
raise requests.exceptions.RequestException( f"Inference retrieval
failed:\n{str(e)}\n{response.content.decode()}" ) if __name__ ==
"__main__": import fire fire.Fire()
```

Best Practices and Tips

1. Enhancing Concept Representation

- Try `finetune_strength` values >1 if the concept is not present
- Increase the `finetune_strength` if the concept is not present or the identity preservation is not strong enough
- Lower the `finetune_strength` if you are unable to generalize the concept into new settings or if the image has visible artifacts

2. Character Training

- Avoid multiple characters in single images
- Use manual captions when multiple characters are unavoidable
- Consider disabling auto-captioning in complex scenes or for complex concepts

3. Quality Considerations

- Use high-quality training images
- Adjust learning rate based on training stability
- Monitor training progress and adjust parameters as needed

4. Prompting

Change the trigger word to something more contextual than "TOK". For example, "mycstm sunglasses" for a product finetune on sunglasses.

Key Strategies:

- **Prepend the trigger word** to your prompt:\ `"mycstm sunglasses, a photo of mycstm sunglasses laying on grass"`
- **For character and product consistency**, briefly describe the person/product:\ `"mycstm man, a photograph of mycstm man, sitting on a park bench and smiling into the camera. mycstm man is a middle aged man with brown curly hair and a beard. He is wearing a cowboy hat."`
- **For styles**, append "in the style of [trigger-word]":\ `"a kangaroo wearing ski goggles holding up a drink, in the style of mycstm watercolourstyle"`

Polling URLs for Finetuned Inference

When running inference on finetuned models, the same polling URL principles apply:

```
python finetuned_inference.py import requests import os import time #
Submit finetuned inference request response = requests.post(
'https://api.us1.bfl.ai/v1/flux-pro-1.1-ultra-finetuned', # Use
appropriate region headers={ 'accept': 'application/json', 'x-key':
os.environ.get("BFL_API_KEY"), 'Content-Type': 'application/json', },
json={ 'prompt': 'A photo of TOK in a serene landscape',
'finetune_id': 'your-finetune-id-here', 'finetune_strength': 1.0,
'aspect_ratio': '16:9' } ) data = response.json() request_id =
data['id'] # For legacy endpoints, construct polling URL manually
polling_url = f"https://api.us1.bfl.ai/v1/get_result" # Poll for
results while True: time.sleep(0.5) result = requests.get(
polling_url, headers={ 'accept': 'application/json', 'x-key':
os.environ.get("BFL_API_KEY"), }, params={'id': request_id} ).json()
if result['status'] == 'Ready': print(f"Finetuned image ready:
{result['result']['sample']}") break elif result['status'] in
['Error', 'Failed']: print(f"Generation failed: {result}") break
```

FLUX Finetuning Guide

The BFL Finetuning API enables you to customize FLUX Pro and FLUX Ultra using 1-20 images of your own visual content, and optionally, text descriptions.

Regional Requirements for Finetuning

Important: Finetuning operations must use region-specific endpoints and cannot use the global or multi-cluster regional endpoints.

Finetuning Endpoints

EU Region: `api.eu1.bfl.ai`

- Use this endpoint for all finetuning operations in the EU region
- Required for GDPR compliance when training with EU data

US Region: `api.us1.bfl.ai`

- Use this endpoint for all finetuning operations in the US region

Regional Consistency

You can only run inference on finetuned models in the same region where they were trained. When you submit a finetuning task using `https://api.us1.bfl.ai/v1/finetune`, you must use `https://api.us1.bfl.ai/v1/flux-pro-1.1-ultra-finetuned` for inference. Cross-region inference will not find your finetuned model.

Getting Started: Step-by-Step Guide

Create a local folder containing your training images: ***Supported formats:*** *JPG, JPEG, PNG, and WebP* **Recommended:** More than 5 images High-quality datasets with clear, articulated subjects/objects/styles significantly improve training results. Higher resolution source images help but are capped at 1MP. Create text files with descriptions for your images: *Text files should share the same name as their corresponding images* **Example:** If your image is `sample.jpg`, create `sample.txt` Compress your folder into a ZIP file containing all images and optional text descriptions. Select appropriate hyperparameters based on your use case. See the Training Parameters section below for detailed configuration options. Use the provided Python functions to submit your finetuning task to the BFL API. Check the status of your training job using the progress monitoring functions. Once training is complete, use your custom model through the available finetuned endpoints.

Model Training Parameters

Required Parameters

Determines the finetuning approach based on your concept. **Options:** `"character"`, `"product"`, `"style"`, `"general"` In "general" mode, the entire image is captioned when captioning is True without specific focus areas. No subject specific improvements will be made. Descriptive note to identify your fine-tune since names are UUIDs. Will be displayed in `finetune_details`.

Optional Parameters

Minimum: 100 Defines training duration. For fast exploration, 100-150 iterations can be enough. For more complex concepts, larger datasets, or extreme precision, more iterations than the default can help. **Default:** 0.00001 if `finetune_type` is "full", 0.0001 if `finetune_type` is "lora" Lower values can improve the result but might need more iterations to learn a concept. Higher values can allow you to train for less iterations at a potential loss in quality. For `finetune_type` "lora", values 10 times larger than for "full" are recommended. **Options:** `"speed"`, `"quality"`, `"high_res_only"` The speed priority will improve speed per training step. Enables/disables automatic image captioning. Unique word/phrase that will be used in the captions to reference the newly introduced concepts. Choose between 32 and 16. A lora_rank of 16 can increase training efficiency and decrease loading times. Choose between "full" for a full finetuning + post hoc extraction of the trained weights into a LoRA or "lora" for a raw LoRA training.

Inference Endpoints

Available endpoints for your finetuned model:

1. `/flux-pro-1.1-ultra-finetuned`
2. `/flux-pro-finetuned`
3. `/flux-pro-1.0-depth-finetuned`
4. `/flux-pro-1.0-canny-finetuned`
5. `/flux-pro-1.0-fill-finetuned`

Regional Inference Requirements

Regional Consistency Required: You can only inference the finetunes in the region that you trained them in. ***EU Finetuned Models:*** *Submit finetuning using:* ***https://api.us1.bfl.ai/v1/finetune****, query using* ***https://api.us1.bfl.ai/v1/flux-pro-1.1-ultra-finetuned*** **US Finetuned Models:** Submit finetuning using: **https://api.eu1.bfl.ai/v1/finetune**, query using **https://api.eu1.bfl.ai/v1/flux-pro-1.1-ultra-finetuned** Querying the global endpoint `https://api.bfl.ai/v1/flux-pro-1.1-ultra-finetuned` will not find your region-specific finetune.

Additional Inference Parameters

The endpoints have additionally all input parameters that their non-finetuned sibling endpoints have. References your specific model. Find the `finetune_id` either in `my_finetunes` or in the return dict of your `/finetune` POST. **Range:** 0-2 Controls finetune influence. Increase this value if your target concept isn't showing up strongly enough. The optimal setting depends on your finetune and prompt.

Implementation Guide

Setup and Dependencies

`bash pip install requests fire` `bash export BFL_API_KEY="your_api_key_here"`

Implementation Guide

Example Python Implementation

`python bfl_finetune.py Assuming you have prepared your images in a `finetuning.zip` file: # submit finetuning task $ python bfl_finetune.py request_finetuning finetuning.zip myfirstfinetune id:`

```
<finetune_id> # query status $ python bfl_finetune.py
finetune_progress <finetune_id> id: <finetune_id> status: Pending
result: null progress: null # once status shows Ready, run inference
(defaults to flux-pro-1.1-ultra-finetuned) $ python bfl_finetune.py
finetune_inference <finetune_id> --prompt="image of a TOK"
finetune_id: <inference_id> # retrieve inference result $ python
bfl_finetune.py get_inference <inference_id> id: <inference_id>
status: Ready result: {"sample": <result_url>, "prompt": "image of a
TOK"} progress: null """ import os import base64 import requests def
request_finetuning( zip_path, finetune_comment, trigger_word="TOK",
mode="general", api_key=None, iterations=300, learning_rate=0.00001,
captioning=True, priority="quality", finetune_type="full",
lora_rank=32, ): """ Request a finetuning using the provided ZIP file.
Args: zip_path (str): Path to the ZIP file containing training data
finetune_comment (str): Comment for the finetune_details trigger_word
(str): Trigger word for the model mode (str): Mode for caption
generation api_key (str): API key for authentication iterations (int):
Number of training iterations learning_rate (float): Learning rate for
optimization captioning (bool): Enable/disable auto-captioning
priority (str): Training quality setting lora_rank (int): Lora rank
finetune_type (str): "full" or "lora" Returns: dict: API response
Raises: FileNotFoundError: If ZIP file is missing
requests.exceptions.RequestException: If API request fails """ if
api_key is None: if "BFL_API_KEY" not in os.environ: raise ValueError(
"Provide your API key via --api_key or an environment variable
BFL_API_KEY" ) api_key = os.environ["BFL_API_KEY"] if not
os.path.exists(zip_path): raise FileNotFoundError(f"ZIP file not found
at {zip_path}") assert mode in ["character", "product", "style",
"general"] with open(zip_path, "rb") as file: encoded_zip =
base64.b64encode(file.read()).decode("utf-8") url =
"https://api.us1.bfl.ai/v1/finetune" headers = { "Content-Type":
"application/json", "X-Key": api_key, } payload = {
"finetune_comment": finetune_comment, "trigger_word": trigger_word,
"file_data": encoded_zip, "iterations": iterations, "mode": mode,
"learning_rate": learning_rate, "captioning": captioning, "priority":
priority, "lora_rank": lora_rank, "finetune_type": finetune_type, }
response = requests.post(url, headers=headers, json=payload) try:
response.raise_for_status() return response.json() except
```

```
requests.exceptions.RequestException as e: raise
requests.exceptions.RequestException( f"Finetune request
failed:\n{str(e)}\n{response.content.decode()}" ) def
finetune_progress( finetune_id, api_key=None, ): if api_key is None:
if "BFL_API_KEY" not in os.environ: raise ValueError( "Provide your
API key via --api_key or an environment variable BFL_API_KEY" )
api_key = os.environ["BFL_API_KEY"] url =
"https://api.us1.bfl.ai/v1/get_result" headers = { "Content-Type":
"application/json", "X-Key": api_key, } payload = { "id": finetune_id,
} response = requests.get(url, headers=headers, params=payload) try:
response.raise_for_status() return response.json() except
requests.exceptions.RequestException as e: raise
requests.exceptions.RequestException( f"Finetune progress
failed:\n{str(e)}\n{response.content.decode()}" ) def finetune_list(
api_key=None, ): if api_key is None: if "BFL_API_KEY" not in
os.environ: raise ValueError( "Provide your API key via --api_key or
an environment variable BFL_API_KEY" ) api_key =
os.environ["BFL_API_KEY"] url =
"https://api.us1.bfl.ai/v1/my_finetunes" headers = { "Content-Type":
"application/json", "X-Key": api_key, } response = requests.get(url,
headers=headers) try: response.raise_for_status() return
response.json() except requests.exceptions.RequestException as e:
raise requests.exceptions.RequestException( f"Finetune listing
failed:\n{str(e)}\n{response.content.decode()}" ) def
finetune_details( finetune_id, api_key=None, ): if api_key is None: if
"BFL_API_KEY" not in os.environ: raise ValueError( "Provide your API
key via --api_key or an environment variable BFL_API_KEY" ) api_key =
os.environ["BFL_API_KEY"] url =
"https://api.us1.bfl.ai/v1/finetune_details" headers = {
"Content-Type": "application/json", "X-Key": api_key, } payload = {
"finetune_id": finetune_id, } response = requests.get(url,
headers=headers, params=payload) try: response.raise_for_status()
return response.json() except requests.exceptions.RequestException as
e: raise requests.exceptions.RequestException( f"Finetune details
failed:\n{str(e)}\n{response.content.decode()}" ) def finetune_delete(
finetune_id, api_key=None, ): if api_key is None: if "BFL_API_KEY" not
in os.environ: raise ValueError( "Provide your API key via --api_key
or an environment variable BFL_API_KEY" ) api_key =
os.environ["BFL_API_KEY"] url =
"https://api.us1.bfl.ai/v1/delete_finetune" headers = {
"Content-Type": "application/json", "X-Key": api_key, } payload = {
"finetune_id": finetune_id, } response = requests.post(url,
headers=headers, json=payload) try: response.raise_for_status() return
response.json() except requests.exceptions.RequestException as e:
raise requests.exceptions.RequestException( f"Finetune deletion
failed:\n{str(e)}\n{response.content.decode()}" ) def
finetune_inference( finetune_id, finetune_strength=1.2,
endpoint="flux-pro-1.1-ultra-finetuned", api_key=None, **kwargs, ): if
api_key is None: if "BFL_API_KEY" not in os.environ: raise ValueError(
"Provide your API key via --api_key or an environment variable
BFL_API_KEY" ) api_key = os.environ["BFL_API_KEY"] url =
f"https://api.us1.bfl.ai/v1/{endpoint}" headers = { "Content-Type":
"application/json", "X-Key": api_key, } payload = { "finetune_id":
finetune_id, "finetune_strength": finetune_strength, **kwargs, }
response = requests.post(url, headers=headers, json=payload) try:
response.raise_for_status() return response.json() except
requests.exceptions.RequestException as e: raise
requests.exceptions.RequestException( f"Finetune inference
failed:\n{str(e)}\n{response.content.decode()}" ) def get_inference(
id, api_key=None, ): if api_key is None: if "BFL_API_KEY" not in
os.environ: raise ValueError( "Provide your API key via --api_key or
an environment variable BFL_API_KEY" ) api_key =
os.environ["BFL_API_KEY"] url = "https://api.us1.bfl.ai/v1/get_result"
headers = { "Content-Type": "application/json", "X-Key": api_key, }
payload = { "id": id, } response = requests.get(url, headers=headers,
params=payload) try: response.raise_for_status() return
response.json() except requests.exceptions.RequestException as e:
raise requests.exceptions.RequestException( f"Inference retrieval
failed:\n{str(e)}\n{response.content.decode()}" ) if __name__ ==
"__main__": import fire fire.Fire()
```

Best Practices and Tips

1. Enhancing Concept Representation

- Try `finetune_strength` values >1 if the concept is not present

- Increase the `finetune_strength` if the concept is not present or the identity preservation is not strong enough
- Lower the `finetune_strength` if you are unable to generalize the concept into new settings or if the image has visible artifacts

2. Character Training

- Avoid multiple characters in single images
- Use manual captions when multiple characters are unavoidable
- Consider disabling auto-captioning in complex scenes or for complex concepts

3. Quality Considerations

- Use high-quality training images
- Adjust learning rate based on training stability
- Monitor training progress and adjust parameters as needed

4. Prompting

Change the trigger word to something more contextual than "TOK". For example, "mycstm sunglasses" for a product finetune on sunglasses.

Key Strategies:

- **Prepend the trigger word** to your prompt:\ `"mycstm sunglasses, a photo of mycstm sunglasses laying on grass"`
- **For character and product consistency**, briefly describe the person/product:\ `"mycstm man, a photograph of mycstm man, sitting on a park bench and smiling into the camera. mycstm man is a middle aged man with brown curly hair and a beard. He is wearing a cowboy hat."`
- **For styles**, append "in the style of [trigger-word]":\ `"a kangaroo wearing ski goggles holding up a drink, in the style of mycstm watercolourstyle"`

Polling URLs for Finetuned Inference

When running inference on finetuned models, the same polling URL principles apply:

```
python finetuned_inference.py import requests import os import time #
Submit finetuned inference request response = requests.post(
'https://api.us1.bfl.ai/v1/flux-pro-1.1-ultra-finetuned', # Use
appropriate region headers={ 'accept': 'application/json', 'x-key':
os.environ.get("BFL_API_KEY"), 'Content-Type': 'application/json', },
json={ 'prompt': 'A photo of TOK in a serene landscape',
'finetune_id': 'your-finetune-id-here', 'finetune_strength': 1.0,
'aspect_ratio': '16:9' } ) data = response.json() request_id =
data['id'] # For legacy endpoints, construct polling URL manually
polling_url = f"https://api.us1.bfl.ai/v1/get_result" # Poll for
results while True: time.sleep(0.5) result = requests.get(
polling_url, headers={ 'accept': 'application/json', 'x-key':
os.environ.get("BFL_API_KEY"), }, params={'id': request_id} ).json()
if result['status'] == 'Ready': print(f"Finetuned image ready:
{result['result']['sample']}") break elif result['status'] in
['Error', 'Failed']: print(f"Generation failed: {result}") break
```

FLUX Finetuning Guide

The BFL Finetuning API enables you to customize FLUX Pro and FLUX Ultra using 1-20 images of your own visual content, and optionally, text descriptions.

Regional Requirements for Finetuning

Important: Finetuning operations must use region-specific endpoints and cannot use the global or multi-cluster regional endpoints.

Finetuning Endpoints

EU Region: `api.eu1.bfl.ai`

- Use this endpoint for all finetuning operations in the EU region
- Required for GDPR compliance when training with EU data

US Region: `api.us1.bfl.ai`

- Use this endpoint for all finetuning operations in the US region

Regional Consistency

You can only run inference on finetuned models in the same region where they were trained. When you submit a finetuning task using `https://api.us1.bfl.ai/v1/finetune`, you must use `https://api.us1.bfl.ai/v1/flux-pro-1.1-ultra-finetuned` for inference. Cross-region inference will not find your finetuned model.

Getting Started: Step-by-Step Guide

Create a local folder containing your training images: ***Supported formats:*** *JPG, JPEG, PNG, and WebP* **Recommended:** More than 5 images High-quality datasets with clear, articulated subjects/objects/styles significantly improve training results. Higher resolution source images help but are capped at 1MP. Create text files with descriptions for your images: *Text files should share the same name as their corresponding images* **Example:** If your image is `sample.jpg`, create `sample.txt` Compress your folder into a ZIP file containing all images and optional text descriptions. Select appropriate hyperparameters based on your use case. See the [Training Parameters] section below for detailed configuration options. Use the provided Python functions to submit your finetuning task to the BFL API. Check the status of your training job using the progress monitoring functions. Once training is complete, use your custom model through the available finetuned endpoints.

Model Training Parameters

Required Parameters

Determines the finetuning approach based on your concept. **Options:** `"character"`, `"product"`, `"style"`, `"general"` In "general" mode, the entire image is captioned when captioning is True without specific focus areas. No subject specific improvements will be made. Descriptive note to identify your fine-tune since names are UUIDs. Will be displayed in `finetune_details`.

Optional Parameters

Minimum: 100 Defines training duration. For fast exploration, 100-150 iterations can be enough. For more complex concepts, larger datasets, or extreme precision, more iterations than the default can help. **Default:** 0.00001 if `finetune_type` is "full", 0.0001 if `finetune_type` is "lora" Lower values can improve the result but might need more iterations to learn a concept. Higher values can allow you to train for less iterations at a potential loss in quality. For `finetune_type` "lora", values 10 times larger than for "full" are recommended. **Options:** `"speed"`, `"quality"`, `"high_res_only"` The speed priority will improve speed per training step. Enables/disables automatic image captioning. Unique word/phrase that will be used in the captions to reference the newly introduced concepts. Choose between 32 and 16. A lora_rank of 16 can increase training efficiency and decrease loading times. Choose between "full" for a full finetuning + post hoc extraction of the trained weights into a LoRA or "lora" for a raw LoRA training.

Inference Endpoints

Available endpoints for your finetuned model:

1. `/flux-pro-1.1-ultra-finetuned`
2. `/flux-pro-finetuned`

3. `/flux-pro-1.0-depth-finetuned`
4. `/flux-pro-1.0-canny-finetuned`
5. `/flux-pro-1.0-fill-finetuned`

Regional Inference Requirements

Regional Consistency Required: You can only inference the finetunes in the region that you trained them in. ***EU Finetuned Models:*** *Submit finetuning using:* ***https://api.us1.bfl.ai/v1/finetune****, query using* ***https://api.us1.bfl.ai/v1/flux-pro-1.1-ultra-finetuned*** **US Finetuned Models:** Submit finetuning using: **https://api.eu1.bfl.ai/v1/finetune**, query using **https://api.eu1.bfl.ai/v1/flux-pro-1.1-ultra-finetuned** Querying the global endpoint `https://api.bfl.ai/v1/flux-pro-1.1-ultra-finetuned` will not find your region-specific finetune.

Additional Inference Parameters

The endpoints have additionally all input parameters that their non-finetuned sibling endpoints have. References your specific model. Find the `finetune_id` either in `my_finetunes` or in the return dict of your `/finetune` POST. **Range:** 0-2 Controls finetune influence. Increase this value if your target concept isn't showing up strongly enough. The optimal setting depends on your finetune and prompt.

Implementation Guide

Setup and Dependencies

```
bash pip install requests fire bash export
BFL_API_KEY="your_api_key_here"
```

Implementation Guide

Example Python Implementation

```
python bfl_finetune.py Assuming you have prepared your images in a
`finetuning.zip` file: # submit finetuning task $ python
bfl_finetune.py request_finetuning finetuning.zip myfirstfinetune id:
<finetune_id> # query status $ python bfl_finetune.py
finetune_progress <finetune_id> id: <finetune_id> status: Pending
result: null progress: null # once status shows Ready, run inference
(defaults to flux-pro-1.1-ultra-finetuned) $ python bfl_finetune.py
finetune_inference <finetune_id> --prompt="image of a TOK"
```

```
finetune_id: <inference_id> # retrieve inference result $ python
bfl_finetune.py get_inference <inference_id> id: <inference_id>
status: Ready result: {"sample": <result_url>, "prompt": "image of a
TOK"} progress: null """ import os import base64 import requests def
request_finetuning( zip_path, finetune_comment, trigger_word="TOK",
mode="general", api_key=None, iterations=300, learning_rate=0.00001,
captioning=True, priority="quality", finetune_type="full",
lora_rank=32, ): """ Request a finetuning using the provided ZIP file.
Args: zip_path (str): Path to the ZIP file containing training data
finetune_comment (str): Comment for the finetune_details trigger_word
(str): Trigger word for the model mode (str): Mode for caption
generation api_key (str): API key for authentication iterations (int):
Number of training iterations learning_rate (float): Learning rate for
optimization captioning (bool): Enable/disable auto-captioning
priority (str): Training quality setting lora_rank (int): Lora rank
finetune_type (str): "full" or "lora" Returns: dict: API response
Raises: FileNotFoundError: If ZIP file is missing
requests.exceptions.RequestException: If API request fails """ if
api_key is None: if "BFL_API_KEY" not in os.environ: raise ValueError(
"Provide your API key via --api_key or an environment variable
BFL_API_KEY" ) api_key = os.environ["BFL_API_KEY"] if not
os.path.exists(zip_path): raise FileNotFoundError(f"ZIP file not found
at {zip_path}") assert mode in ["character", "product", "style",
"general"] with open(zip_path, "rb") as file: encoded_zip =
base64.b64encode(file.read()).decode("utf-8") url =
"https://api.us1.bfl.ai/v1/finetune" headers = { "Content-Type":
"application/json", "X-Key": api_key, } payload = {
"finetune_comment": finetune_comment, "trigger_word": trigger_word,
"file_data": encoded_zip, "iterations": iterations, "mode": mode,
"learning_rate": learning_rate, "captioning": captioning, "priority":
priority, "lora_rank": lora_rank, "finetune_type": finetune_type, }
response = requests.post(url, headers=headers, json=payload) try:
response.raise_for_status() return response.json() except
requests.exceptions.RequestException as e: raise
requests.exceptions.RequestException( f"Finetune request
failed:\n{str(e)}\n{response.content.decode()}" ) def
finetune_progress( finetune_id, api_key=None, ): if api_key is None:
if "BFL_API_KEY" not in os.environ: raise ValueError( "Provide your
API key via --api_key or an environment variable BFL_API_KEY" )
api_key = os.environ["BFL_API_KEY"] url =
"https://api.us1.bfl.ai/v1/get_result" headers = { "Content-Type":
"application/json", "X-Key": api_key, } payload = { "id": finetune_id,
} response = requests.get(url, headers=headers, params=payload) try:
response.raise_for_status() return response.json() except
requests.exceptions.RequestException as e: raise
requests.exceptions.RequestException( f"Finetune progress
failed:\n{str(e)}\n{response.content.decode()}" ) def finetune_list(
api_key=None, ): if api_key is None: if "BFL_API_KEY" not in
os.environ: raise ValueError( "Provide your API key via --api_key or
an environment variable BFL_API_KEY" ) api_key =
os.environ["BFL_API_KEY"] url =
"https://api.us1.bfl.ai/v1/my_finetunes" headers = { "Content-Type":
"application/json", "X-Key": api_key, } response = requests.get(url,
headers=headers) try: response.raise_for_status() return
response.json() except requests.exceptions.RequestException as e:
raise requests.exceptions.RequestException( f"Finetune listing
failed:\n{str(e)}\n{response.content.decode()}" ) def
finetune_details( finetune_id, api_key=None, ): if api_key is None: if
"BFL_API_KEY" not in os.environ: raise ValueError( "Provide your API
key via --api_key or an environment variable BFL_API_KEY" ) api_key =
os.environ["BFL_API_KEY"] url =
"https://api.us1.bfl.ai/v1/finetune_details" headers = {
"Content-Type": "application/json", "X-Key": api_key, } payload = {
"finetune_id": finetune_id, } response = requests.get(url,
headers=headers, params=payload) try: response.raise_for_status()
return response.json() except requests.exceptions.RequestException as
e: raise requests.exceptions.RequestException( f"Finetune details
failed:\n{str(e)}\n{response.content.decode()}" ) def finetune_delete(
finetune_id, api_key=None, ): if api_key is None: if "BFL_API_KEY" not
in os.environ: raise ValueError( "Provide your API key via --api_key
or an environment variable BFL_API_KEY" ) api_key =
os.environ["BFL_API_KEY"] url =
"https://api.us1.bfl.ai/v1/delete_finetune" headers = {
"Content-Type": "application/json", "X-Key": api_key, } payload = {
"finetune_id": finetune_id, } response = requests.post(url,
headers=headers, json=payload) try: response.raise_for_status() return
```

```
response.json() except requests.exceptions.RequestException as e:
raise requests.exceptions.RequestException( f"Finetune deletion
failed:\n{str(e)}\n{response.content.decode()}" ) def
finetune_inference( finetune_id, finetune_strength=1.2,
endpoint="flux-pro-1.1-ultra-finetuned", api_key=None, **kwargs, ): if
api_key is None: if "BFL_API_KEY" not in os.environ: raise ValueError(
"Provide your API key via --api_key or an environment variable
BFL_API_KEY" ) api_key = os.environ["BFL_API_KEY"] url =
f"https://api.us1.bfl.ai/v1/{endpoint}" headers = { "Content-Type":
"application/json", "X-Key": api_key, } payload = { "finetune_id":
finetune_id, "finetune_strength": finetune_strength, **kwargs, }
response = requests.post(url, headers=headers, json=payload) try:
response.raise_for_status() return response.json() except
requests.exceptions.RequestException as e: raise
requests.exceptions.RequestException( f"Finetune inference
failed:\n{str(e)}\n{response.content.decode()}" ) def get_inference(
id, api_key=None, ): if api_key is None: if "BFL_API_KEY" not in
os.environ: raise ValueError( "Provide your API key via --api_key or
an environment variable BFL_API_KEY" ) api_key =
os.environ["BFL_API_KEY"] url = "https://api.us1.bfl.ai/v1/get_result"
headers = { "Content-Type": "application/json", "X-Key": api_key, }
payload = { "id": id, } response = requests.get(url, headers=headers,
params=payload) try: response.raise_for_status() return
response.json() except requests.exceptions.RequestException as e:
raise requests.exceptions.RequestException( f"Inference retrieval
failed:\n{str(e)}\n{response.content.decode()}" ) if __name__ ==
"__main__": import fire fire.Fire()
```

Best Practices and Tips

1. Enhancing Concept Representation

- Try `finetune_strength` values >1 if the concept is not present
- Increase the `finetune_strength` if the concept is not present or the identity preservation is not strong enough
- Lower the `finetune_strength` if you are unable to generalize the concept into new settings or if the image has visible artifacts

2. Character Training

- Avoid multiple characters in single images
- Use manual captions when multiple characters are unavoidable
- Consider disabling auto-captioning in complex scenes or for complex concepts

3. Quality Considerations

- Use high-quality training images
- Adjust learning rate based on training stability
- Monitor training progress and adjust parameters as needed

4. Prompting

Change the trigger word to something more contextual than "TOK". For example, "mycstm sunglasses" for a product finetune on sunglasses.

Key Strategies:

- **Prepend the trigger word** to your prompt:\ `"mycstm sunglasses, a photo of mycstm sunglasses laying on grass"`
- **For character and product consistency**, briefly describe the person/product:\ `"mycstm man, a photograph of mycstm man, sitting on a park bench and smiling into the camera. mycstm man is a middle aged man with brown curly hair and a beard. He is wearing a cowboy hat."`
- **For styles**, append "in the style of [trigger-word]":\ `"a kangaroo wearing ski goggles holding up a drink, in the style of mycstm watercolourstyle"`

Polling URLs for Finetuned Inference

When running inference on finetuned models, the same polling URL principles apply:

```
python finetuned_inference.py import requests import os import time #
Submit finetuned inference request response = requests.post(
'https://api.us1.bfl.ai/v1/flux-pro-1.1-ultra-finetuned', # Use
appropriate region headers={ 'accept': 'application/json', 'x-key':
os.environ.get("BFL_API_KEY"), 'Content-Type': 'application/json', },
json={ 'prompt': 'A photo of TOK in a serene landscape',
'finetune_id': 'your-finetune-id-here', 'finetune_strength': 1.0,
'aspect_ratio': '16:9' } ) data = response.json() request_id =
data['id'] # For legacy endpoints, construct polling URL manually
polling_url = f"https://api.us1.bfl.ai/v1/get_result" # Poll for
results while True: time.sleep(0.5) result = requests.get(
polling_url, headers={ 'accept': 'application/json', 'x-key':
os.environ.get("BFL_API_KEY"), }, params={'id': request_id} ).json()
if result['status'] == 'Ready': print(f"Finetuned image ready:
{result['result']['sample']}") break elif result['status'] in
['Error', 'Failed']: print(f"Generation failed: {result}") break
```

Get Result

An endpoint for getting generation task result.

OpenAPI

````yaml https://api.bfl.ai/openapi.json get /v1/get_result paths: path: /v1/get_result method: get servers:

```
- url: https://api.bfl.ai
  description: BFL API
```

request: security: [] parameters: path: {} query: id: schema:

```
    - type: string
      required: true
      title: Id
  header: {}
  cookie: {}
body: {}
```

response: '200': application/json: schemaArray:

```
  - type: object
    properties:
      id:
        allOf:
          - type: string
            title: Id
            description: Task id for retrieving result
      status:
        allOf:
          - $ref: '#/components/schemas/StatusResponse'
      result:
```
````

```
          allOf:
            - anyOf:
                - {}
                - type: 'null'
              title: Result
        progress:
          allOf:
            - anyOf:
                - type: number
                - type: 'null'
              title: Progress
        details:
          allOf:
            - anyOf:
                - type: object
                - type: 'null'
              title: Details
      title: ResultResponse
      refIdentifier: '#/components/schemas/ResultResponse'
      requiredProperties:
        - id
        - status
  examples:
    example:
      value:
        id: <string>
        status: Task not found
        result: <any>
        progress: 123
        details: {}
  description: Successful Response
'422':
  application/json:
    schemaArray:
      - type: object
        properties:
          detail:
            allOf:
              - items:
                  $ref: '#/components/schemas/ValidationError'
                type: array
                title: Detail
        title: HTTPValidationError
        refIdentifier: '#/components/schemas/HTTPValidationError'
    examples:
      example:
        value:
          detail:
            - loc:
                - <string>
              msg: <string>
              type: <string>
    description: Validation Error
```

deprecated: false type: path components: schemas: StatusResponse: type: string enum:

```
   - Task not found
    - Pending
    - Request Moderated
    - Content Moderated
    - Ready
    - Error
  title: StatusResponse
ValidationError:
  properties:
    loc:
      items:
        anyOf:
          - type: string
          - type: integer
      type: array
      title: Location
    msg:
      type: string
      title: Message
    type:
      type: string
      title: Error Type
  type: object
  required:
    - loc
    - msg
    - type
  title: ValidationError
```

# Get Result

> An endpoint for getting generation task result.

## OpenAPI

````yaml https://api.bfl.ai/openapi.json get /v1/get_result
paths:
  path: /v1/get_result
  method: get
  servers:
    - url: https://api.bfl.ai
      description: BFL API
  request:
    security: []
    parameters:
      path: {}
      query:
        id:
          schema:
            - type: string
              required: true
              title: Id
      header: {}
      cookie: {}
    body: {}
  response:
    '200':
      application/json:
        schemaArray:
          - type: object
            properties:
              id:
                allOf:
                  - type: string
                    title: Id
                    description: Task id for retrieving result
              status:
                allOf:
                  - $ref: '#/components/schemas/StatusResponse'
              result:
                allOf:
                  - anyOf:
                      - {}
                      - type: 'null'
````

```
              title: Result
          progress:
            allOf:
              - anyOf:
                  - type: number
                  - type: 'null'
                title: Progress
          details:
            allOf:
              - anyOf:
                  - type: object
                  - type: 'null'
                title: Details
        title: ResultResponse
        refIdentifier: '#/components/schemas/ResultResponse'
        requiredProperties:
          - id
          - status
    examples:
      example:
        value:
          id: <string>
          status: Task not found
          result: <any>
          progress: 123
          details: {}
    description: Successful Response
'422':
  application/json:
    schemaArray:
      - type: object
        properties:
          detail:
            allOf:
              - items:
                  $ref: '#/components/schemas/ValidationError'
                type: array
                title: Detail
        title: HTTPValidationError
        refIdentifier: '#/components/schemas/HTTPValidationError'
    examples:
      example:
        value:
          detail:
                - loc:
                    - <string>
                  msg: <string>
                  type: <string>
        description: Validation Error
  deprecated: false
  type: path
components:
  schemas:
    StatusResponse:
      type: string
      enum:
        - Task not found
        - Pending
        - Request Moderated
        - Content Moderated
        - Ready
        - Error
      title: StatusResponse
    ValidationError:
      properties:
        loc:
          items:
            anyOf:
              - type: string
              - type: integer
          type: array
          title: Location
        msg:
          type: string
          title: Message
        type:
          type: string
          title: Error Type
      type: object
      required:
        - loc
        - msg
        - type
      title: ValidationError
```

My Finetunes

List all finetune_ids created by the user

OpenAPI

````yaml https://api.bfl.ai/openapi.json get /v1/my_finetunes paths: path: /v1/my_finetunes method: get servers:

```
- url: https://api.us1.bfl.ai
  description: BFL Finetune API
```

request: security:

```
 - title: APIKeyHeader
    parameters:
      query: {}
      header:
        x-key:
          type: apiKey
      cookie: {}
parameters:
  path: {}
  query: {}
  header: {}
  cookie: {}
body: {}
```

response: '200': application/json: schemaArray:

```
     - type: object
        properties:
          finetunes:
            allOf:
              - items: {}
                type: array
                title: Finetunes
                description: List of finetunes created by the user
        title: MyFinetunesResponse
        refIdentifier: '#/components/schemas/MyFinetunesResponse'
        requiredProperties:
          - finetunes
    examples:
      example:
```
````

```
        value:
          finetunes:
            - <any>
    description: Successful Response
```

deprecated: false type: path components: schemas: {}

# Delete Finetune

> Delete a finetune_id that was created by the user

## OpenAPI

````yaml https://api.bfl.ai/openapi.json post /v1/delete_finetune
paths:
  path: /v1/delete_finetune
  method: post
  servers:
    - url: https://api.us1.bfl.ai
      description: BFL Finetune API
  request:
    security:
      - title: APIKeyHeader
        parameters:
          query: {}
          header:
            x-key:
              type: apiKey
          cookie: {}
    parameters:
      path: {}
      query: {}
      header: {}
      cookie: {}
    body:
      application/json:
        schemaArray:
          - type: object
            properties:
              finetune_id:
                allOf:
                  - type: string
````

```
                title: Finetune Id
                description: ID of the fine-tuned model you want to delete.
                example: my-finetune
            required: true
            title: DeleteFinetuneInputs
            refIdentifier: '#/components/schemas/DeleteFinetuneInputs'
            requiredProperties:
              - finetune_id
        examples:
          example:
            value:
              finetune_id: my-finetune
  response:
    '200':
      application/json:
        schemaArray:
          - type: object
            properties:
              status:
                allOf:
                  - type: string
                    title: Status
                    description: Status of the deletion
              message:
                allOf:
                  - type: string
                    title: Message
                    description: Message about the deletion
              deleted_finetune_id:
                allOf:
                  - type: string
                    title: Deleted Finetune Id
                    description: ID of the deleted finetune
              timestamp:
                allOf:
                  - type: string
                    title: Timestamp
                    description: Timestamp of the deletion
            title: DeleteFinetuneResponse
            refIdentifier: '#/components/schemas/DeleteFinetuneResponse'
            requiredProperties:
              - status
              - message
              - deleted_finetune_id
```

```yaml
              - timestamp
        examples:
          example:
            value:
              status: <string>
              message: <string>
              deleted_finetune_id: <string>
              timestamp: <string>
        description: Successful Response
    '422':
      application/json:
        schemaArray:
          - type: object
            properties:
              detail:
                allOf:
                  - items:
                      $ref: '#/components/schemas/ValidationError'
                    type: array
                    title: Detail
            title: HTTPValidationError
            refIdentifier: '#/components/schemas/HTTPValidationError'
        examples:
          example:
            value:
              detail:
                - loc:
                    - <string>
                  msg: <string>
                  type: <string>
        description: Validation Error
  deprecated: false
  type: path
components:
  schemas:
    ValidationError:
      properties:
        loc:
          items:
            anyOf:
              - type: string
              - type: integer
          type: array
          title: Location
        msg:
          type: string
          title: Message
        type:
          type: string
          title: Error Type
      type: object
      required:
        - loc
        - msg
        - type
      title: ValidationError
```

Delete Finetune

> Delete a finetune_id that was created by the user

OpenAPI

````yaml https://api.bfl.ai/openapi.json post /v1/delete_finetune paths: path: /v1/delete_finetune method: post servers:

```
- url: https://api.us1.bfl.ai
  description: BFL Finetune API
```

request: security:

```
 - title: APIKeyHeader
    parameters:
      query: {}
      header:
        x-key:
          type: apiKey
      cookie: {}
parameters:
  path: {}
  query: {}
  header: {}
  cookie: {}
body:
  application/json:
```
````

```
    schemaArray:
      - type: object
        properties:
          finetune_id:
            allOf:
              - type: string
                title: Finetune Id
                description: ID of the fine-tuned model you want to delete.
                example: my-finetune
        required: true
        title: DeleteFinetuneInputs
        refIdentifier: '#/components/schemas/DeleteFinetuneInputs'
        requiredProperties:
          - finetune_id
    examples:
      example:
        value:
          finetune_id: my-finetune
```

response: '200': application/json: schemaArray:

```
     - type: object
        properties:
          status:
            allOf:
              - type: string
                title: Status
                description: Status of the deletion
          message:
            allOf:
              - type: string
                title: Message
                description: Message about the deletion
          deleted_finetune_id:
            allOf:
              - type: string
                title: Deleted Finetune Id
                description: ID of the deleted finetune
          timestamp:
            allOf:
              - type: string
                title: Timestamp
                description: Timestamp of the deletion
        title: DeleteFinetuneResponse
        refIdentifier: '#/components/schemas/DeleteFinetuneResponse'
        requiredProperties:
          - status
          - message
          - deleted_finetune_id
          - timestamp
    examples:
      example:
        value:
          status: <string>
          message: <string>
          deleted_finetune_id: <string>
          timestamp: <string>
    description: Successful Response
'422':
  application/json:
    schemaArray:
      - type: object
        properties:
          detail:
            allOf:
              - items:
                  $ref: '#/components/schemas/ValidationError'
                type: array
                title: Detail
        title: HTTPValidationError
        refIdentifier: '#/components/schemas/HTTPValidationError'
    examples:
      example:
        value:
          detail:
            - loc:
                - <string>
              msg: <string>
              type: <string>
    description: Validation Error
```

deprecated: false type: path components: schemas: ValidationError: properties: loc: items: anyOf:

```
         - type: string
          - type: integer
      type: array
      title: Location
```

```yaml
    msg:
      type: string
      title: Message
    type:
      type: string
      title: Error Type
  type: object
  required:
    - loc
    - msg
    - type
  title: ValidationError
```

# Delete Finetune

> Delete a finetune_id that was created by the user

## OpenAPI

````yaml https://api.bfl.ai/openapi.json post /v1/delete_finetune
paths:
  path: /v1/delete_finetune
  method: post
  servers:
    - url: https://api.us1.bfl.ai
      description: BFL Finetune API
  request:
    security:
      - title: APIKeyHeader
        parameters:
          query: {}
          header:
            x-key:
              type: apiKey
          cookie: {}
    parameters:
      path: {}
      query: {}
      header: {}
      cookie: {}
    body:
      application/json:
````

```
      schemaArray:
        - type: object
          properties:
            finetune_id:
              allOf:
                - type: string
                  title: Finetune Id
                  description: ID of the fine-tuned model you want to delete.
                  example: my-finetune
          required: true
          title: DeleteFinetuneInputs
          refIdentifier: '#/components/schemas/DeleteFinetuneInputs'
          requiredProperties:
            - finetune_id
      examples:
        example:
          value:
            finetune_id: my-finetune
response:
  '200':
    application/json:
      schemaArray:
        - type: object
          properties:
            status:
              allOf:
                - type: string
                  title: Status
                  description: Status of the deletion
            message:
              allOf:
                - type: string
                  title: Message
                  description: Message about the deletion
            deleted_finetune_id:
              allOf:
                - type: string
                  title: Deleted Finetune Id
                  description: ID of the deleted finetune
            timestamp:
              allOf:
                - type: string
                  title: Timestamp
                  description: Timestamp of the deletion
          title: DeleteFinetuneResponse
          refIdentifier: '#/components/schemas/DeleteFinetuneResponse'
          requiredProperties:
            - status
            - message
            - deleted_finetune_id
            - timestamp
        examples:
          example:
            value:
              status: <string>
              message: <string>
              deleted_finetune_id: <string>
              timestamp: <string>
        description: Successful Response
    '422':
      application/json:
        schemaArray:
          - type: object
            properties:
              detail:
                allOf:
                  - items:
                      $ref: '#/components/schemas/ValidationError'
                    type: array
                    title: Detail
            title: HTTPValidationError
            refIdentifier: '#/components/schemas/HTTPValidationError'
        examples:
          example:
            value:
              detail:
                - loc:
                    - <string>
                  msg: <string>
                  type: <string>
        description: Validation Error
  deprecated: false
  type: path
components:
  schemas:
    ValidationError:
      properties:
        loc:
```

```
    items:
      anyOf:
        - type: string
        - type: integer
    type: array
    title: Location
  msg:
    type: string
    title: Message
  type:
    type: string
    title: Error Type
type: object
required:
  - loc
  - msg
  - type
title: ValidationError
```