ChatGPT

Building a “Forensic-Style” Face–Text Dataset: Data Collection & Preprocessing

1. Survey of Existing Face Datasets

Public Face Benchmarks: Modern face datasets vary in size, image quality, diversity, and usage rights. **Table 1** compares three popular benchmarks:

Dataset	**Size** (images)	**Resolution**	**Diversity & Content**	**Licensing / Ethics**
FFHQ (Flickr-Faces-HQ)	70,000	1024×1024 PNG	High variation in age, ethnicity, and accessories (eyeglasses, hats, etc.) [1]; images are well-aligned and centered faces	Sourced from Flickr under permissive CC licenses; Non-commercial use (CC BY-NC-SA); filtered to remove artwork and non-human faces via Mechanical Turk.
CelebA / CelebA-HQ	202,599 (CelebA); 30,000 HQ subset	~178×218 (CelebA); 1024×1024 (HQ)	Images of ~10k celebrities with varied poses and background clutter; CelebA-HQ is a higher-quality subset. Each image has **40 annotated attributes** (e.g. *Smiling*, *Male*, *Wearing Hat*). Diversity is moderate (mostly Western celebs, so demographic balance is limited).	Available for research only (non-commercial). Images were obtained online without explicit subject consent (public figures), raising some privacy considerations.
VGGFace2	3.31 million	Varying (typically high-res)	9,131 identities with an average of 362 images each. Includes a wide range of poses, ages, ethnicities, and professions (actors, athletes, politicians). Collected via Google Image Search with automated + manual cleaning to ensure correct identity labeling.	Provided for research; many images are of public figures. Usage is subject to license terms (no commercial FR use). The large scale and “in the wild” sourcing mean potential bias (e.g. more images of famous or Western individuals) but efforts were made to increase diversity.

Law-Enforcement & Forensic Datasets: Beyond academic benchmarks, there are specialized face sets used in forensic or security contexts: - **Mugshot & Longitudinal Databases:** *MORPH* is a longitudinal mugshot dataset with 55,134 images of 13,617 subjects (age 16–77). It reflects real arrest photos (useful for age progression and ID) but is proprietary and not openly licensed. Such mugshot datasets often over-represent certain demographics (e.g. minorities) and raise ethical issues since subjects did not consent to public AI use. - **IJB Series (IJB-A/B/C):** NIST's "IARPA Janus" benchmarks contain thousands of law-enforcement style photos (including surveillance frames) for face recognition evaluation. These are more representative of forensic scenarios (uncooperative subjects, varied lighting), but access is restricted and images are sensitive. Privacy and consent are concerns – these sets are used under controlled agreements. - **Composite Sketch Databases:** For training models to generate faces from descriptions, some datasets pair **forensic sketches** with photos. E.g., the *UoM-SGFS* (Univ. of Malta – Software-Generated Face Sketch) dataset contains a large number of computer-generated composite sketches designed to mimic police identikit sketches. Such datasets are valuable for sketch-to-photo matching research, but sketches lack the detail of real photos. **CUFSF** and **IIIT-D Sketch** are other examples pairing hand-drawn or computer-generated sketches with face images. These are typically smaller scale and intended for recognition algorithm testing rather than generative modeling. - **Surveillance Footage Datasets:** Sets like *SCFace* (surveillance camera face dataset) contain low-quality images of subjects at various distances, reflecting CCTV scenarios. These help algorithms handle blur/low-res, but often have limited subjects (SCFace: 130 subjects).

Synthetic & Composite Face Datasets: Synthetic data offers an appealing alternative when privacy is a concern: - **Face Synthetics (Microsoft, 2021):** 100,000 photorealistic faces generated via a parametric 3D model and rendering pipeline. Comes with **landmarks and segmentation maps** for each face. Models trained on this purely synthetic set have shown good generalization to real faces. **Pros:** No privacy issues (no real people) and labels (landmarks, etc.) are perfectly accurate by construction. Highly diverse in pose, lighting, expression by design. **Cons:** Might still miss subtle real-world imperfections; requires a complex generation pipeline to ensure realism. - **100K Fake Faces (StyleGAN/Stable Diffusion):** Researchers have released sets of StyleGAN-generated faces (e.g. **100K-Faces** by Nvidia) and Stable-Diffusion-generated faces. These typically are high-resolution and diverse if the generator was trained on a broad dataset. **Pros:** No consent needed; can be generated in huge quantities. **Cons:** Quality depends on the generative model (earlier GAN faces might have artifacts). Also, if the generator's training data had biases, those carry into the fakes. - **Synthetic Composite Descriptions:** Some projects generate **text–face pairs** synthetically. For example, the *Face2Text* dataset has ~400 real face images with 1,400 textual descriptions, and one can augment it with synthetic data. A 2019 project combined Face2Text with CelebA attributes by auto-generating text from attributes. This yielded ~197k images × 3 descriptions each (≈591k pairs) after filtering. Synthetic description generation (using a template and vocabulary) ensured a large paired dataset, though the sentences may sound formulaic.

Pros/Cons Summary: Real face datasets (FFHQ, CelebA, VGGFace2) provide authentic variation and high fidelity, but come with **privacy and bias concerns**. Law enforcement sets offer domain-specific data but are hard to obtain and ethically fraught (e.g. lack of consent, skewed demographics). **Synthetic data** can fill gaps, providing *bias-controlled*, license-free images – for instance, the SFHQ dataset released 425k high-quality synthetic faces with no privacy or copyright issues. However, synthetic faces must be carefully validated for realism, and a model trained only on fakes might still need fine-tuning on real images to capture subtleties. In practice, a **hybrid approach** (both real and synthetic) is beneficial, leveraging the abundance of synthetic data while retaining a core of real images for realism. (Indeed, the user suggests using **both** data types for maximal benefit.)

2. Annotation & Description Standards

Facial Attribute Taxonomy: To describe faces consistently, it's useful to define a taxonomy of facial attributes. Key categories include:

- **Basic Demographics:** Age group (infant, child, teenager, young adult, middle-aged, elderly), Gender presentation (male, female, non-binary – though many legacy datasets use binary M/F).
- **Facial Shape & Structure:** Face shape (oval, round, square, heart-shaped) [2], jawline (strong, narrow), cheekbones (high, low).
- **Hair:** Hair color (blonde, brown, black, red, gray) [2], hair length (short, medium, long), texture (straight, wavy, curly, coily), hairstyle (bald/shaved, bangs/fringe, parted). Facial hair for men: beard, mustache, goatee (or clean-shaven).
- **Eyes & Brows:** Eye color (brown, blue, green, hazel, etc.), eye shape (round, almond, narrow) – though shape is harder to label consistently. Presence of **eyewear** (eyeglasses or sunglasses). Eyebrow style: thick/bushy, thin, arched.
- **Nose & Mouth:** Nose size/shape (e.g. aquiline, button nose, "pointy" nose). Mouth/lips (full lips vs. thin lips, mouth open or closed). **Teeth** visible or not (smiling broadly vs. neutral).
- **Skin & Complexion:** Skin tone (light, medium, dark – often best described using comparative terms rather than race labels), presence of freckles or wrinkles, facial hair coverage. Ethnicity or racial category can be included (**with care**): some datasets use broad groups (e.g. East Asian, South Asian, Black, White, Latino, Middle Eastern), but **we recommend describing observable features** (skin tone, etc.) rather than impute ethnicity which can be error-prone and sensitive.
- **Expression:** Neutral, smiling (grinning, smirking), frowning, surprised (mouth open, raised eyebrows), angry (frown, tense jaw), etc. E.g. CelebA has a binary *Smiling* attribute.
- **Accessories & Occlusions:** Hats, caps, or headwear; jewelry like earrings or necklace; scarf or mask (if any); headphones; **makeup** (yes/no, heavy/light – e.g. CelebA's *Heavy Makeup* attribute). Occlusions like **eyeglasses** (regular or sunglasses), or objects partly blocking the face need noting.
- **Distinctive Features:** Scars, birthmarks, moles, tattoos on face; dimples; facial markings. If relevant to forensic use, also note "facial hair style" (e.g. stubble, sideburns) and hairline (receding hairline, widow's peak).

It's important to pre-define these attributes with *clear definitions*. For instance, what counts as "arched eyebrows" or "pointy nose"? A consistent rubric helps annotators agree.

CelebA's 40 attributes touch many of the above (e.g. *Bald, Bangs, Black Hair, Blond Hair, Brown Hair, Bushy Eyebrows, Arched Eyebrows, Narrow Eyes, Big Nose, Pointy Nose, Oval Face, Pale Skin, Rosy Cheeks, Male, Young, Wearing Hat, Eyeglasses, Mustache,* etc.). The **Police Officers Faces (POF)** dataset provides an even more fine-grained schema with **73 attributes** grouped into Age, Gender, and visual traits like hair color, face shape, presence of makeup [3].

Structured vs. Free-Form Descriptions: We have two main approaches to describing faces in text:

- *Structured annotations* – a fixed set of attributes or tags. **Pros:** Consistency and easy to parse. Attributes can be binary or multi-class (e.g. hair_color = {blond, brown, black, red, gray}). Datasets like CelebA rely on binary labels for attributes, which can be programmatically converted to sentences ("She has blond hair, no eyeglasses, wearing lipstick, and is smiling"). Controlled vocabularies ensure that the same terms are used by everyone (avoiding one annotator saying

"spectacles" vs another "glasses"). **Cons:** Structured labels might miss nuance (e.g. *pointy nose* = *TRUE/FALSE* ignores degrees of pointiness) and can be incomplete (they don't capture an overall impression or unusual features not in the schema).
- *Free-form descriptions* – e.g. full-sentence captions of a face. Example: "A young man with short black hair and thick eyebrows, smiling and wearing a police uniform." **Pros:** Rich detail combining attributes (age, hair, expression, attire) in natural language. Can mention unique features ("scar on left cheek") that structured labels might not include. This is useful for training text-to-image models to handle unconstrained inputs. **Cons:** Variability in wording – two annotators may describe the same face differently. Free text is harder to analyze automatically; requires NLP processing. Inconsistent vocabulary or grammar can confuse the model (one might say "elderly man" vs "old man" vs "70-year-old male"). Ensuring quality is challenging.

In practice, **a hybrid approach** can work well: use a structured attribute template to guarantee key details, but allow free-form elaboration for additional description. For example, *CelebA-Dialog (2021)* provides each image with both the 40 binary attributes and a natural-language caption describing the face. This gives the best of both worlds – the model can learn from precise labels and from fluent sentences.

Annotation Workflows & Quality Control: Creating a forensic-style dataset likely involves human annotators describing faces from images (or from memory, in a true forensic scenario). Best practices include:

- **Guidelines & Training:** Provide annotators with a well-defined attribute list and example descriptions. Include instructions on sensitive categories (e.g. avoid guessing someone's race or name; describe observable traits only).
- **Controlled Vocabulary:** Use a *defined set of terms* for certain attributes to avoid ambiguity. For instance, decide on a standard set of hair colors ("blond" vs "blonde" spelled consistently; use "gray" instead of "silver" for hair, etc.). This avoids a messy vocabulary later. One can maintain a glossary of allowed terms for accessories, colors, etc.
- **Multiple Annotators & Consensus:** To ensure consistency, have multiple annotators describe some overlapping subset of images and measure inter-annotator agreement. High agreement indicates reliable attribute definitions. Notably, a study on CelebA found that for only 12 of the 40 attributes did two independent annotators exceed 95% agreement; attributes like *High Cheekbones, Pointy Nose,* and *Oval Face* had essentially random agreement. This underscores how subjective some traits are, and why careful definition and possibly training/calibration of annotators is needed. If annotators disagree frequently (low Cohen's κ), you may need to refine instructions or provide more examples.
- **Inter-annotator Agreement Metrics:** Track metrics such as Cohen's Kappa or % agreement on a validation set of images annotated by all. Set a threshold (e.g. $\geq$0.8 Kappa) for acceptable agreement on each attribute. If certain labels like "Attractive = Yes/No" are too subjective (CelebA had an *Attractive* attribute, which is inherently subjective), consider dropping or redefining those.
- **Quality Audits:** Use automated checks to catch errors. For instance, run a face detector on all images and ensure a face is present and unobstructed for each (one researcher did this to filter out images where the face was not clear). If an image has *Eyeglasses=TRUE* but the face detector finds no eyes (maybe the person turned away), flag it for review. Likewise, if free-form description says "woman" but the structured label says *Male*, that's a discrepancy to resolve.
- **Consistency in Free-Form Text:** Enforce basic rules in descriptions: e.g. always mention gender and approximate age early in the sentence, then notable features. You might supply a template:

"<Approx. Age>, <Gender> with <hair description>, <facial feature>, ...". This ensures each description covers key facets in a predictable order, aiding both annotators and the model.
- **Spell-check and Grammar:** Run a spell-check on descriptions (tools like spaCy or LanguageTool can help) to catch typos that could confuse the tokenizer. Controlled vocabulary also helps here (fewer synonyms = fewer chances for misspelling).
- **Ethical labeling:** Decide upfront how to handle sensitive attributes. For example, **race/ethnicity**: If included, it should be coarse and for a purpose (e.g. to ensure diversity in generation). Alternatively, stick to describing skin tone or origin clues (accent would be audio, so N/A here). Avoid derogatory terms or subjective judgments (no "ugly" or "attractive" labels – those are problematic). **Consent in annotation:** If using crowdworkers, ensure they're informed about the purpose (especially since forensic imagery can be sensitive).

By following these standards, we create a dataset where each face image is paired with a rich, accurate textual description and/or a set of attribute labels. This provides the grounding needed to train a latent diffusion model to generate faces from textual cues.

3. Image Preprocessing Best Practices

Raw face images collected "in the wild" will vary in size, background, alignment, etc. Preprocessing aims to standardize the images while preserving the identity and key features of each face.

3.1 Face Detection & Alignment: The first step is to detect the face and align it. Alignment typically means adjusting the image so that the eyes and facial landmarks are in consistent positions (e.g. eyes horizontal at a fixed y-coordinate). This was done for FFHQ using **dlib**'s 68-point facial landmark detector. Common alignment procedure: - Detect landmarks (eyes corners, nose tip, mouth corners, etc.) using a tool like dlib, **MTCNN** (Multi-Task Cascaded CNN), or **MediaPipe Face Mesh**. - Compute an affine transform to fix rotation and scale – e.g. rotate the image so the line connecting the eyes is horizontal, and scale/crop such that the distance between the eyes is constant in all images. - Crop the image to a square region around the face. Many face datasets crop roughly to include the entire head and some hair, but little background. For example, ArcFace (a face recognition model) uses aligned 112×112 crops obtained via 5 landmarks (centers of eyes, nose base, mouth corners). Similarly, FFHQ images are centered and cropped to contain the full head with a bit of shoulders/background.

Alignment ensures that the model doesn't have to learn face position from scratch – all faces are centered and upright. **However,** over-aggressive alignment (e.g. warping profile faces to front) is not desired – we still want natural pose variety. Usually only in-plane rotation and uniform scaling are applied (no 3D pose normalization).

3.2 Resize to 512×512: Latent diffusion models often train on 256×256 or 512×512 images. We choose 512 here for high quality. After alignment and cropping to a square, resize each image to 512×512 pixels. Use a high-quality resampling (Lanczos filter) to minimize blur. This resolution balances detail with memory – it's the standard used in Stable Diffusion v1 (trained on 512px images).

3.3 Color Normalization: Ensure images are in a consistent color format (e.g. sRGB). It's common to normalize pixel values for model input: for example, scale pixel intensities to **[0,1]** or **[-1,1]**. With PyTorch, one might convert to tensor and then apply `transforms.Normalize(mean=[0.5,0.5,0.5], std=[0.5,0.5,0.5])` so that each channel is in [-1,1]. This speeds up training convergence. Also consider

gamma correction if images come from different sources – but if all are standard photographs, this is likely consistent.

3.4 Handling Multiple Faces: If an image contains more than one face, decide on a policy. For a forensic dataset, typically each image is a single individual. If not, you may need to either crop out each face as separate images or discard images with multiple people to avoid ambiguity in the text description ("two men" would complicate generation). Most curated datasets (CelebA, VGGFace2) either have one face per image or provide bounding boxes for each face. For simplicity, we focus on one face per image.

3.5 Trade-offs – Identity vs. Framing: We must preserve the person's identity-critical features (relative geometry of eyes, nose, mouth, etc.) while achieving a consistent framing. Slight misalignment can hurt the model's ability to learn fine details (like angle of face). On the other hand, cropping too loosely includes background clutter; cropping too tightly might cut off context like hairstyle. A common compromise: crop so that ~80% of the 512×512 frame is the face region, and include the hair and some shoulders. This keeps the *identity* (facial details) clear while still showing context that might be described (hair style, earrings). **Identity preservation** also means avoiding any image operations that distort the face: e.g. no aspect ratio stretching. Always keep aspect ratio until the final square crop (which should be chosen carefully around the face). If your source images are all frontal and similarly composed (like mugshots), heavy alignment might not be needed; but if not, alignment greatly helps consistency.

3.6 Tools and Libraries: Many open-source tools can automate these steps: - *Face alignment & crop:* **face_recognition** (Python library) can detect and crop faces easily. **InsightFace** provides cutting-edge detectors (RetinaFace) and alignment utilities. Dlib's `align_face` script (used in FFHQ) is available and widely used. - *Image processing:* Python PIL or OpenCV for resizing and cropping. E.g. `img = img.resize((512,512), PIL.Image.LANCZOS)`. Libraries like **albumentations** or **torchvision.transforms** also have resize, crop, normalize functions that can be composed.

Pitfalls: be cautious about **image artifacts** – e.g. some images might have watermarks or compression artifacts. You might want to detect and remove or mask watermarks (so the model doesn't learn text artifacts). Also, if some images are grayscale and others color, convert grayscale to RGB (duplicate channels) for consistency. Finally, perform a manual spot-check: look at a grid of preprocessed images to ensure faces are not cut off or overly zoomed. A few mis-cropped images (e.g. if the detector failed and we cropped the background) could confuse the model, so better to weed those out.

By standardizing image inputs (aligned, scaled, normalized), we reduce unwanted variability and ensure the model focuses on *facial features* relevant to generation, rather than irrelevant differences in scale or orientation.

4. Text Preprocessing Pipeline

The textual side of the dataset – whether structured labels or free-form descriptions – also needs cleaning and normalization before feeding into a model or tokenizer.

4.1 Cleaning & Normalization: Start by cleaning the raw text annotations: - **Lowercasing:** Convert all text to lowercase (unless case carries meaning, which it usually doesn't for face descriptions). This ensures consistency (e.g. "Brown hair" vs "brown hair" treated the same). Most NLP pipelines do this. - **Remove**

punctuation that is not needed. Commas and periods can be helpful in sentences, but extraneous punctuation or special characters should be removed. For instance, strip out any quotes, semicolons, or stray characters. Ensure no control characters or HTML entities remain if data came from web. - **Spell-check and typo correction:** Run a spell-checker or dictionary lookup. If workers wrote descriptions, there could be misspellings ("brwn eyes" -> "brown eyes"). Tools like **spaCy** or **TextBlob** can help identify likely misspelled words. For critical attributes, you can enforce the controlled vocabulary (replace slang or misspellings with the standard term). - **Expand any abbreviations:** e.g. if someone wrote "blk hair", convert to "black hair". - **Remove redundant whitespace:** Strip leading/trailing spaces, condense multiple spaces to one.

After these steps, each description should be a clean, well-formed sentence or list of attributes. For example, `"She has curly blond hair , and blue eyes!!"` would become `"she has curly blond hair and blue eyes"`.

4.2 Tokenization Strategy: Latent diffusion models typically rely on a tokenizer from a language model (e.g. CLIP's BPE tokenizer or a BERT tokenizer) to convert text to embeddings. You have a few options: - Use an **off-the-shelf tokenizer** from a pretrained model (recommended if you'll use that model's text encoder, like CLIP or BERT). For instance, Stable Diffusion v1 uses CLIP's tokenizer (based on byte-pair encoding) – this will handle casing (it lowercases internally) and punctuation elegantly. Hugging Face's `AutoTokenizer` can load a pretrained tokenizer easily:

```
from transformers import AutoTokenizer
tokenizer = AutoTokenizer.from_pretrained("openai/clip-vit-large-patch14")
tokens = tokenizer.encode(text, truncation=True, max_length=77)
```

This yields a list of token IDs ready for input to the model's text encoder. - Alternatively, for a custom model pipeline, you might use a simpler tokenizer (word-level or char-level). For example, the Char-CNN-RNN text encoder used in some GAN research doesn't require a predefined vocab. But modern practice is to use BPE/WordPiece due to efficiency and the ability to handle any word (by subword decomposition).

Preprocessing for tokenization means ensuring the text fits the tokenizer's requirements. Since we already lowercased and removed weird characters, the main thing is to decide on **max length**. If descriptions average, say, 15 words, a max length of 50 tokens is plenty. We might truncate longer captions or split them if a dataset provides multiple sentences per image. (CelebA-Dialog, for instance, has both a caption and an editing request; one could concatenate or choose one.)

4.3 Handling Structured Labels in Text Form: If your dataset has structured attributes but you want to feed them as text (to utilize a text encoder), you can convert the attributes into a sentence. This was done by the aforementioned project: they took CelebA's 40 binary attributes and generated sentences like "He has brown hair and is smiling while wearing glasses". Writing a simple template generator with some randomness (to avoid all sentences being identical) is useful. For example, create templates like: - `A ${age_group} ${gender} with ${hair_color} ${hair_type} hair and ${misc_features}...` Then fill in with the attribute values for each image. Introducing synonyms or slight variations in templates (as was done with a vocabulary of ~205 words and random order choice) can prevent the model from overfitting to a single sentence structure.

If multiple descriptions per image are available, each can be a separate training pair (image, text).

4.4 JSON/CSV Schema Design: You need a way to store and later ingest the face-text pairs. Two common formats: - **CSV**: each line has an image filename and either a single description or multiple attribute columns. E.g., a CSV with columns: `image_id, description`. This is simple but if you have multiple descriptions per image, you'd repeat the image ID on multiple lines. - **JSON**: a structured JSON is more flexible. For instance:

```
[
  {
    "image": "face_00001.jpg",
    "attributes": {
       "age": "young adult", "gender": "male", "hair_color": "black", ...
    },
    "descriptions": [
       "a young man with short black hair and high cheekbones, smiling
slightly",
       "young male, black hair, oval face, no facial hair, looks happy"
    ]
  },
  ...
]
```

This JSON array allows multiple descriptions and also stores the raw attribute labels for reference. During training, you could either use the full sentence from `descriptions` or even concatenate the structured labels into a pseudo-sentence.

Make sure to escape any special characters in JSON and validate it (no trailing commas, proper quotes, etc.). Many libraries (Python's `json` or pandas) can help output the data properly.

4.5 Off-the-shelf NLP Tools: A few useful libraries for preprocessing: - **spaCy:** Fast and easy for tokenization, POS tagging, etc. It can be used to tokenize or to verify parts of speech in descriptions (for example, ensure at least one adjective and noun are present – indicating the description has a descriptor and a noun like "woman"). - **NLTK:** Has classic tokenizers and stopword lists. You might not need stopword removal here (since function words like "with" or "and" are fine in descriptions), but NLTK's word_tokenize could split text if you needed custom processing. - **HuggingFace Tokenizers:** If performance is a concern (very large text data), the tokenizers library is very fast for BPE tokenization. But for 600k captions or so, Python-based is fine. - **LanguageTool or Ginger:** for grammar/spell checking if needed.

Deduplication: If any identical descriptions are repeated (especially if auto-generated from attributes, you might get many repeated sentences), consider deduplicating or at least diversifying them. Too many identical text examples can bias the model. In the CelebA attribute-to-text approach, they generated 3 descriptions per image with slight variation to mitigate this. You could do similar or just ensure a variety of wording across the dataset.

Finally, ensure the **text and image IDs are properly linked**. A common pitfall is a misalignment between image order and caption order. Using unique IDs in a JSON or including the image filename in each row (for CSV) prevents confusion.

By cleaning and structuring the text data, we prepare it for efficient ingestion by the model's text encoder. The end result is that for each image we have a tokenized, normalized description ready to be paired with the image during training.

5. Dataset Splitting & Stratification

When the dataset is prepared, we split it into training, validation, and test sets (commonly 80/10/10%). The key is to do this **stratified**, so that each split is balanced across important attributes. This prevents, say, all elderly faces ending up in the test set, or one gender dominating the train set.

5.1 Identify Stratification Factors: Decide which attributes you need to balance. Likely candidates: - **Gender**, **Age group**, **Ethnicity/Race**, **Key appearance features** (hair color, presence of glasses, etc.). You may also consider *identity* if multiple images per person – usually we ensure identities are separated between train/val/test for face recognition tasks. For generation, it's less clear-cut, but to test model generalization, you might set aside some identities or attribute combinations.

5.2 Stratified Shuffle Split: We can use scikit-learn's utilities to split while preserving class distributions. The challenge: we have multiple attributes to balance simultaneously (multivariate stratification). A common solution is to create a **composite stratification key**. For example, combine key attributes into one label string for stratification:

```
import pandas as pd
from sklearn.model_selection import StratifiedShuffleSplit

# df has columns: gender, age_group, ethnicity, hair_color, etc.
# Create a composite label for stratification (e.g. Gender-AgeGroup)
df['stratify_label'] = df['gender'] + "_" + df['age_group']
```

If finer stratification is needed, you can include more attributes (but combining too many can lead to very many unique combos). Perhaps combine gender, broad age (child/adult/senior), and ethnicity into one label. The goal is that each such combo appears in all splits in similar proportions.

Using scikit-learn:

```
splitter = StratifiedShuffleSplit(n_splits=1, test_size=0.10, random_state=42)
for train_idx, test_idx in splitter.split(df, df['stratify_label']):
    train_df = df.iloc[train_idx]
    temp_df  = df.iloc[test_idx]
```

This gives a 90% train, 10% temp (which we'll later split into val/test). Then we can stratify-split `temp_df` (or use `train_test_split` with stratify) to get val and test, stratifying on the same label to maintain distribution.

If using pandas or just Python lists, it's conceptually similar: group by the composite key and sample proportionally.

5.3 Handling Multi-Label Stratification: The above composite works if you have a few broad categories. If you need to balance many attributes at once, consider an **iterative stratification** algorithm (there are libraries like `iterative-stratification` for multi-label). But for our face dataset, focusing on a handful of demographic attributes is usually enough. For example, ensure gender and ethnicity are balanced (50/50 gender in each split, and roughly the same ethnic composition in each). Age can be trickier if continuous; you might bucket ages into ranges (0-18, 18-30, 30-50, 50+).

5.4 Code Example: Below is a conceptual example using scikit-learn's `train_test_split` with stratification on multiple variables by zipping them (stratify on a tuple of (gender, age_group)):

```
from sklearn.model_selection import train_test_split
# Assume we have lists or Series: images, descriptions, gender_labels,
age_labels
# Create stratify keys as tuple of (gender, age_group) for each sample
stratify_keys = list(zip(gender_labels, age_labels))
X_train, X_temp, y_train, y_temp = train_test_split(images, descriptions,
                                                    test_size=0.2,
                                                    stratify=stratify_keys,
                                                    random_state=42)
# Now X_temp (20% of data) will be split into val and test 10% each:
gender_temp = [g for g, a in zip(gender_labels, age_labels) if ...]  #
corresponding temp genders
age_temp = [a for g, a in zip(gender_labels, age_labels) if ...]
stratify_temp = list(zip(gender_temp, age_temp))
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp,
                                                test_size=0.5,
                                                stratify=stratify_temp,
                                                random_state=42)
```

This ensures each subset (train/val/test) has, for example, the same ~50% male, ~30% young adult, etc. as the full set.

5.5 Identity Splitting: If the dataset has multiple images of the same person (e.g. VGGFace2 style), consider keeping all images of a given identity in **one** split (especially if you plan to evaluate identity consistency). For generation, it might not be critical, but for fairness you might want to see if the model can generate faces of people it hasn't seen in training (conceptually). If needed, stratify by identity: e.g. ensure each identity is assigned wholly to train or test, not both (this is more relevant for recognition tasks, but worth noting).

5.6 Verify Stratification: After splitting, it's good to verify the attribute distribution. Calculate the percentage of each key attribute in each split and compare to original. For example, if original dataset was 40% female, 60% male, each split should be close to that (within a few tenths of a percent ideally). You can tabulate age group counts, etc., to ensure balance:

```
for subset, name in [(train_df, "Train"), (val_df, "Val"), (test_df, "Test")]:
    print(name, "gender dist:", subset['gender'].value_counts(normalize=True))
    print(name, "ethnicity dist:",
subset['ethnicity'].value_counts(normalize=True))
```

All should match the overall distribution.

Stratification prevents evaluation biases. For instance, without it, one could end up with **val/test containing only, say, young female Asian faces**, and the model might perform poorly simply because it never trained on that group – giving a false sense of failure. With stratified splits, each subset is a microcosm of the whole dataset's diversity.

In summary, use stratified splitting to maintain representativeness. Tools like scikit-learn's `StratifiedShuffleSplit` make it easy to split on one label; for multiple, combine labels or use a grouping approach. This results in train/val/test sets that are **balanced and fair** for downstream model training and evaluation.

6. Augmentation Guidelines

Data augmentation can bolster the model's robustness by providing variations of the images. For a face generation task, we want "light" augmentations that don't fundamentally alter the person's identity or described attributes.

6.1 Recommended Augmentations:

- **Horizontal Flip:** Flipping an image left-right is a common augmentation in face datasets (essentially mirroring the face). It doubles the data and helps the model not become "left-right biased." For example, a face looking left vs right should be treated similarly by the model. Flips are used in virtually all face recognition training pipelines. **Caution:** If textual descriptions include asymmetric details ("scar on left cheek"), a flip would contradict the text. In a controlled dataset, we might avoid referencing left/right in descriptions, or if present, skip flipping those cases. If using flips, ensure either descriptions are generic enough or consider generating a new description for the flipped image (swapping left/right in text, which is complex). In many cases, forensic descriptions may not mention left/right, focusing on the feature itself, so flips are usually safe. We apply horizontal flip with probability $p = 0.5$ to training images.
- **Color Jitter:** Small random changes to brightness, contrast, saturation, and hue. This simulates different lighting conditions and camera exposures. For instance, randomly make an image slightly brighter or more washed out. This helps the model not latch onto specific lighting in the training images. A typical range is ±10-20% for brightness/contrast, and slight shifts in color balance. E.g. use

an augmentation like *ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2, hue=0.1)*. **Caution:** Too much jitter (like huge hue shifts) could change perceived hair or skin color; keep it subtle.

- **Slight Rotation:** Small rotations (e.g. −10° to +10°) can help if the model should handle slightly tilted faces. Since we aligned faces, this augmentation reintroduces minor pose variation. It helps because during generation, we might want the ability to tilt the head. Don't over-rotate (a 90° rotated face is unrealistic and not described in text). Keep within ~±15°. Use *rotate* with `fill` (to fill corners) or crop after rotate.
- **Scaling/Cropping:** Randomly zoom in or out a bit (say, ±5% zoom) and crop to 512×512. This simulates distances. It can also counter any "exact framing" bias from perfect alignment. Albumentations or torchvision's *RandomResizedCrop(512, scale=(0.95, 1.0))* can implement this.
- **Gaussian Noise or Blur:** A tiny amount of Gaussian noise or a slight blur can improve robustness to image quality. For instance, add noise with σ=0.01 of pixel range, or apply a blur with a small kernel occasionally. This reflects variations in camera focus. But use sparingly (e.g. 10% of images augmented with a blur).
- **Random Occlusion (Light):** In some forensic scenarios, part of a face might be covered (scarf, etc.). We can simulate minor occlusions: e.g. *Cutout* augmentation which blacks out a small random square on the image. However, occluding key features could confuse the model if the text doesn't mention it. Perhaps limit to occluding corners or background. This augmentation is less common for generation tasks but mentioned for completeness.

It's generally **not** recommended to do vertical flips (upside-down faces) – that's not a realistic appearance and would break our text description assumptions. Also avoid heavy warps or elastic distortions which change facial structure.

6.2 Tools for Augmentation: - **Albumentations** is a powerful library with many of these transforms. For example:

```
import albumentations as A
transform = A.Compose([
    A.HorizontalFlip(p=0.5),
    A.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2, hue=0.1, p=0.5),
    A.Rotate(limit=10, border_mode=0, value=(0,0,0), p=0.5),
    A.OneOf([
       A.GaussianBlur(blur_limit=3, p=0.2),
       A.GaussNoise(var_limit=(10.0, 50.0), p=0.2)
    ], p=0.3)
])
augmented = transform(image=image)
```

This would randomly apply flips, color jitter, slight rotation, and occasionally blur or noise. - **torchvision.transforms** for simpler pipelines: e.g. `transforms.RandomHorizontalFlip()`, `transforms.ColorJitter()`, etc., can be composed. - **imgaug** or **OpenCV** can also do these.

6.3 Parameter Tuning: Keep augmentation parameters mild. The goal is to simulate realistic variation: - Horizontal flip: yes (since human faces are roughly symmetric, a flip still looks like a plausible new individual most times). - Brightness/contrast: ±20% is usually fine – any more and you might turn day into night or

wash out skin tones. - Saturation/hue: small shifts so that, for example, a red shirt doesn't become green (unless we explicitly want clothes color variation – but that would diverge from any text description of clothing). - Rotation: limit to angles where a person might tilt their head naturally (few degrees). - Crop: do not cut off important parts (avoid cropping out the chin or forehead entirely). - Noise/blur: low magnitude, just to simulate camera noise or slight out-of-focus.

6.4 Effect on Identity: We must ensure these augmentations don't confuse the model about identity or attributes. Flips can actually *change* a perceived attribute if asymmetrical (like a parted hairstyle going from left to right), but since descriptions likely won't mention part side, it's fine. Color jitter could slightly alter hair color perception (e.g. very light blond vs white in extreme cases), hence keeping it small. Overall, these augmentations are chosen because they **don't alter the fundamental facial geometry**, so the face still represents the same person with the same features.

6.5 Augmenting Text? While the question focuses on image augmentation, note that one can also augment text descriptions in some cases (paraphrasing). E.g. using a thesaurus to swap some words ("wearing a hat" vs "with a hat on"). This can increase variety. But automated paraphrasing can introduce errors, so it must be done carefully or via templates as mentioned.

In summary, apply augmentations that improve robustness to flips and lighting. Empirically, for face tasks, **horizontal flipping** is one of the most effective augmentations (doubles data, improves recognition and likely generation coverage). Other augments give diminishing returns but can help cover edge cases. Always ensure augmented images still match their text descriptions in essence. It's wise to **disable augmentations for the validation/test sets** (you want to evaluate on the original data distribution). Augmentation is only a training-time trick to enrich the dataset the model "sees."

7. Ethical, Legal, and Privacy Considerations

Building and using a face-text dataset raises serious ethical and legal questions. We must address these proactively:

7.1 Consent and Privacy: Face images are biometric data. In many jurisdictions (e.g. EU GDPR, California CCPA, Illinois BIPA) biometric data is sensitive personal information. *Whenever possible, obtain consent* from individuals whose faces are used. This could mean using photos released under licenses that imply consent for reuse (e.g. FFHQ using Flickr images under Creative Commons) – though even CC licenses don't equate to consent for facial recognition uses. Avoid scraping images of private individuals without permission. The MS-Celeb-1M dataset is a famous cautionary tale: it amassed 10 million images of ~100k people from the internet and was later **taken down** after criticism that many individuals (not actual celebrities in many cases) were included without consent, potentially running afoul of GDPR. Microsoft removed the dataset quietly in 2019 once the ethical and legal issues were exposed.

For a forensic-style dataset, if using mugshots or police data, note that those individuals did not consent to their images being used in AI training. This could be legally restricted (police departments usually cannot freely share such data due to privacy laws or department policies). Always anonymize any personal identifiers (names, IDs). If images are from public domain sources (like old police bulletins), double-check the usage rights.

7.2 Anonymization Strategies: If you must include real individuals, consider techniques to **anonymize** or at least reduce risk: - Remove or do not collect any metadata linking the photo to a real identity (no names, no social media handles, etc., in the dataset). - If possible, apply face de-identification (like slight feature blending). However, that conflicts with maintaining realism for a generation task. A compromise could be using synthetic faces (no real identity at all) for the majority of data. - Provide an **opt-out mechanism**: If someone finds their image in the dataset (e.g. if dataset is public), they should be able to request removal. Clearly document how to contact and honor such requests swiftly. - Consider only using images of **public figures** or actors who have a multitude of public images (as CelebA did). While not true "consent," the argument is these images are already public and the persons are accustomed to their images being widely seen. However, even this is not legally bulletproof – e.g. in the EU, a public figure still has privacy rights and rights to their own data usage.

7.3 Data Protection Compliance: If distributing the dataset or using it commercially, ensure compliance: - **GDPR (EU):** Requires a lawful basis for processing personal data. For research, one might use "legitimate interests" or get explicit consent. GDPR also gives individuals the right to deletion ("right to be forgotten"), which is problematic once data is in a trained model. Techniques like not storing raw images or using synthetic data can mitigate this. - **CCPA (California):** Grants consumers rights to know and delete personal info collected. If any California residents are in the data, theoretically they could demand removal. - **Illinois BIPA:** Prohibits collecting/using biometric info without written consent. A face dataset of Illinois residents without consent could trigger liability (this law is why companies like Clearview AI faced lawsuits). - **Global variances:** Some countries ban certain AI uses or have strict privacy (e.g. China has rules about deepfakes and data security, which might impact a face generation model usage).

Given these complexities, a strong recommendation is to **favor datasets that avoid personal data altogether.** Using **synthetic faces sidesteps privacy issues entirely**. If synthetic data is used, state clearly that "no real person's image is included; all faces are AI-generated" – this alleviates privacy/legal concerns and can be a selling point.

7.4 Face Recognition & Misuse: A forensic face-text model might be used to generate a suspect's face from a description. This borders on face recognition technology, which is controversial. Ethical considerations: - **Bias and Fairness:** If the dataset isn't diverse, the model could generate poorer quality or stereotyped faces for underrepresented groups. We must ensure the training data is balanced (as discussed) and evaluate outputs for bias. For example, does the model systematically lighten skin or change features when given certain prompts? These issues should be tested and documented. - **Misidentification risk:** In a forensic context, an AI-generated face could be mistakenly identified as a real person (a false lead). It's crucial that any such system is used as an aid, not as sole evidence. This is more about model deployment, but dataset bias contributes to this risk. - **Deepfake and Impersonation:** A powerful face generator can be misused to create fake identities or impersonate people. While our purpose is presumably lawful (forensic reconstruction), one must consider safeguards. Possibly *watermark* the generated images or restrict the model so it's only used with oversight. If the dataset included real faces of people, there's also a risk of overfitting and regenerating someone's actual face – hence another argument for synthetic or many individuals to dilute any single person's influence.

7.5 Licensing the Dataset: Decide on a dataset license that reflects these ethical guardrails. Non-commercial research license (like FFHQ's CC-BY-NC) can prevent commercial exploitation. Include an **AI Data Sheet / Documentation** detailing: - Source of images and annotations. - Demographics distribution and any bias known. - Any individuals who opted out or were removed. - Intended use (e.g. "not for facial

recognition or surveillance" if that's a concern). - Limitations: e.g. "not validated for biometric identification; for research on generation only."

7.6 Example – IBM Diversity in Faces: IBM released a "Diversity in Faces" dataset of 1 million images with detailed annotations to improve fairness. They used publicly available Flickr images (under licenses), but even that drew criticism as photographers consented to sharing but subjects didn't necessarily consent to biometric analysis. Transparency and allowing opt-out were key issues raised. The lesson: even well-intentioned datasets need careful handling of consent.

In conclusion, prioritize **privacy by design**: if possible, use generated or fully public-domain data. If real faces are involved, get consent or stick to those who have implicitly given it (e.g. celebrities in public photos, understanding the limitations). Obey relevant laws – e.g., avoid using any EU citizen images without GDPR compliance, avoid Illinois residents' images without consent, etc. Implement an opt-out and document your process. And finally, consider the downstream effects: clearly warn users of the dataset/model that it can't be used to identify real persons (to avoid turning it into a surveillance tool). As the POF project highlighted, facial recognition tech can reinforce harmful biases – we must ensure our dataset does not contribute to that by being as fair, open, and respectful of rights as possible.

Deliverables

High-Level Data Collection & Preprocessing Pipeline

Pipeline Overview: The process of building the dataset can be visualized in sequential steps:

```
[Raw Image Sources] + [Raw Text/Attribute Annotations]
            │
            ├─► **Data Annotation** (if needed): Label facial attributes; write
textual descriptions for each face (using controlled vocabulary & guidelines).
            │
            ├─► **Quality Check**: Remove bad data (faces not detected, images
too low-res, inconsistent annotations).
            │
            ├─► **Image Preprocessing**: Face detection → alignment (rotate/
scale) → crop to face → resize to 512×512 → normalize pixels.
            │             (Tools: dlib, OpenCV, etc.)
            │
            ├─► **Text Preprocessing**: Clean text (lowercase, remove typos/
punctuation) → tokenize or encode text (using CLIP/BERT tokenizer) → prepare
description strings.
            │             (Tools: spaCy, HuggingFace Transformers)
            │
            ├─► **Augmentation (training only)**: Apply on-the-fly
augmentations to images (flip, color jitter, etc.) to enrich training data. (No
augmentations on validation/test images.)
```

```
            │
            └─▶ **Dataset Split & Export**: Stratified shuffle split into
Train/Val/Test ensuring balanced demographics. Save images and accompanying
captions/attributes in a JSON or CSV.
```

Each step feeds into the next, resulting in a ready-to-use dataset for model training.

Implementation Steps (Bullet List with Code Snippets)

1. **Collect & Annotate Data:** Obtain face images from chosen sources (e.g. FFHQ for open-source images, or your own collected photos). If not already annotated with text:
2. Create an annotation interface (could be a simple web tool or spreadsheet) for human labelers to describe each face.
3. Ensure guidelines are provided (attribute taxonomy, example descriptions).
4. *Output:* a file (CSV/JSON) of image filenames with their descriptions/attributes.
5. **Verify Face Detection:** Run a face detector to ensure each image actually contains a face (and possibly only one face).

   ```
   import cv2
   face_cascade = cv2.CascadeClassifier('haarcascade_frontalface.xml')
   for img_path in image_list:
       img = cv2.imread(img_path)
       faces = face_cascade.detectMultiScale(img, 1.3, 5)
       if len(faces) == 0:
           print("No face detected in", img_path)
           # Decide to remove or fix this image
   ```

   Remove images with no (or too many) faces. This script uses a simple Haar cascade; modern detectors like MTCNN or RetinaFace can be used for better accuracy.

6. **Face Alignment & Cropping:** For each image, detect landmarks and align:

   ```
   import face_alignment  # e.g. using face_alignment library
   fa = face_alignment.FaceAlignment(face_alignment.LandmarksType._2D,
   flip_input=False)
   for img_path in image_list:
       img = cv2.imread(img_path)[:,:,::-1]  # BGR to RGB
       preds = fa.get_landmarks(img)
       if preds:
           # Compute transform from preds[0] (landmarks) to reference
   landmarks
           aligned_img = align_face(img, preds[0])  # custom function or use
   ```

```
library
        cv2.imwrite("aligned/"+img_path, aligned_img[:,:,::-1])
```

(This is pseudo-code; in practice one might use dlib's `align` *function or similar. FFHQ's* `align_images.py` *script is a reference.)*

7. **Resize & Normalize:** After alignment, ensure image is 512×512:

```
from PIL import Image
img = Image.open(aligned_path)
img = img.resize((512,512), Image.LANCZOS)
img.save(resized_path, format='PNG')
```

Normalization (scaling pixel values) will be done later when loading for training (e.g. via PyTorch DataLoader transforms), so the saved images remain in standard 8-bit format.

8. **Text Cleaning:** Clean the descriptions programmatically:

```
import re, string
def clean_text(desc):
    desc = desc.lower().strip()
    desc = re.sub(r'\s+', ' ', desc)               # collapse whitespace
    desc = desc.translate(str.maketrans('', '', string.punctuation))  #
remove punctuation
    return desc

df['description_clean'] = df['description'].apply(clean_text)
```

Review a few cleaned outputs to ensure it's as expected (e.g., "bluegreen eyes" might become "bluegreen eyes" – which might actually need a space inserted, so refine rules as needed).

9. **Tokenization Example:** Using Hugging Face tokenizer (for CLIP or other model):

```
from transformers import CLIPTokenizer
tokenizer = CLIPTokenizer.from_pretrained("openai/clip-vit-large-patch14")
# Assume df is a DataFrame with a 'description_clean' column
df['tokens'] = df['description_clean'].apply(lambda x: tokenizer.encode(x,
truncation=True, max_length=77))
```

This adds a list of token IDs for each description. If using your own model, you might skip this and just work with the text, letting the DataLoader tokenize on the fly.

10. **Prepare Final Data Structures:** Decide format – e.g. store everything in a JSON for easy loading:

```
import json
records = []
for i, row in df.iterrows():
    rec = {
      "image": row['image_filename'],
      "description": row['description_clean']
    }
    # If structured attributes are available, include them
    rec["attributes"] = {attr: row[attr] for attr in
["age_group","gender","hair_color",...]}
    records.append(rec)
with open("face_text_dataset.json", "w") as f:
    json.dump(records, f, indent=2)
```

This JSON list can be read by training code to get images and text.

11. **Stratified Splitting:** Use `StratifiedShuffleSplit` to split indices by, say, gender and age:

```
from sklearn.model_selection import StratifiedShuffleSplit
strat_labels = df['gender'] + "_" + df['age_group']
splitter = StratifiedShuffleSplit(n_splits=1, test_size=0.2,
random_state=42)
train_idx, test_idx = next(splitter.split(df, strat_labels))
train_df = df.iloc[train_idx]
test_df_full = df.iloc[test_idx]
# Further split test_df_full into val and test (10% each of original)
similarly
```

This yields `train_df`, `val_df`, `test_df`. Save these splits, e.g. as separate JSON files or add a "split" field in the records.

12. **Augmentation Setup:** For training, configure augmentation in your data pipeline (not applied offline to saved files, but at training time). For example, in PyTorch:

```
import torchvision.transforms as T
train_transforms = T.Compose([
    T.RandomHorizontalFlip(p=0.5),
    T.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2, hue=0.05),
    T.ToTensor(),
    T.Normalize([0.5,0.5,0.5], [0.5,0.5,0.5])
])
```

The DataLoader would apply `train_transforms` on the fly. Validation/test would use just `T.ToTensor()` and `T.Normalize(...)` without random transforms. Libraries like

Albumentations can be integrated via a custom transform function if using PyTorch or used in a TensorFlow data pipeline similarly.

13. **Final Checks:** Verify a small batch from `train_df`:
    - Load an image, apply transforms, and decode a token sequence to text (inverse tokenization) to confirm everything lines up.
    - Check that, for instance, a face with “wearing sunglasses” actually had sunglasses in the image, etc. (This can be done manually for a handful or via a simple attribute classifier as a secondary validation).

Now your dataset is ready for model training. You would feed batches of (image_tensor, token_tensor) into the latent diffusion model training routine.

Next-Steps Checklist to Validate Dataset Readiness

- [ ] **Face Detection & Alignment Verified:** All images have a properly centered face of the correct person. No images are missing or showing the wrong content (e.g. background only). Misaligned or poor-quality images have been removed or fixed.
- [ ] **Attribute-Text Consistency:** The textual descriptions or labels match each image. (E.g., if description says "wearing glasses," the image indeed has glasses [1].) Run a quick consistency script or manual sample check.
- [ ] **Balanced Splits:** Training, validation, and test sets have similar distribution of genders, ages, ethnicities, etc.. No subset is missing a critical demographic that is present in others.
- [ ] **No Sensitive Identifiers:** The dataset contains no names or IDs of real people (unless absolutely necessary and consented). Any metadata EXIF tags in images have been stripped. Individuals cannot be easily re-identified from the data (especially if public).
- [ ] **Consent & License Documentation:** Confirm that all data usage is under proper license or consent. Document the sources of images and their licenses in a README. (For synthetic data, explicitly note no real persons are included.)
- [ ] **Augmentation Parameters Tuned:** The chosen augmentations have been tested to ensure they don’t break the data-label alignment (e.g., flipping doesn’t create wrong descriptions). Augmentations are applied only in training pipeline, not permanently to files.
- [ ] **Tokenization Verified:** Ensure that the text tokenizer can handle all description texts (no unknown tokens issues). The max token length is set such that no important part of description is truncated.
- [ ] **Data Loading Performance:** If the dataset is large, test that images can be read from disk fast enough (consider storing images in a LMDB or using caching if needed). Verify that the JSON/CSV reading and parsing is correct.
- [ ] **Documentation Complete:** Create a datasheet/model card for the dataset covering its composition, intended use, and limitations. Include the ethical considerations and disclaimer about not using the dataset for unauthorized surveillance or identification.
- [ ] **Sample Throughput Test:** Do a dry-run where you simulate one epoch of training (or a few batches) to ensure everything flows without errors. Monitor memory usage with 512×512 images and adjust batch size if needed.

By ticking off each item above, you can be confident that the dataset is ready and that you've done due diligence on quality, fairness, and legality. The resulting dataset will be a solid foundation for training a latent diffusion model to generate "forensic-style" face images from textual descriptions.

---

1 GitHub - NVlabs/ffhq-dataset: Flickr-Faces-HQ Dataset (FFHQ)
https://github.com/NVlabs/ffhq-dataset

2 3 Police Officers Faces - ASRG
https://algorithmic-sabotage.github.io/asrg/police_officers-faces-pof/