ChatGPT

Offline AI Solutions for Cosmetic Facial Analysis

Plastic surgeons seeking an **offline, image-only AI pipeline** for facial analysis have a range of open-source solutions to consider. Key tasks include **facial landmark detection**, **feature extraction**, and **classification** of cosmetic issues (e.g. asymmetry, disproportion, or features commonly targeted in surgery). Below we compare top approaches that meet the requirements (fully local on a GTX 4090, accuracy-over-speed, and mostly open-source). We then detail implementation steps for the two best solutions, including model sources, licenses, and hardware needs. Throughout, we note how each system can improve over time with surgeon feedback.

Comparison of Top Open-Source Approaches

1. 2D Landmark-Based Deep Learning Pipeline

This approach uses high-accuracy facial landmark detection and classic anthropometric analysis, optionally enhanced with deep learning classifiers for specific features. It typically involves: **(a)** detecting the face, **(b)** locating key facial landmarks, **(c)** measuring symmetry/proportions from those landmarks, and **(d)** classifying cosmetic issues based on those measurements or via learned models.

- **Face & Landmark Detection:** State-of-the-art open models like **RetinaFace** (for detection) and **FAN (Face Alignment Network)** or **MediaPipe Face Mesh** (for landmarks) can be used. RetinaFace is widely regarded as the most accurate open-source face detector [1], providing reliable face bounding boxes. For landmark localization, deep networks achieve extremely precise results – e.g. the **face-alignment** library (Bulat et al.) uses a CNN to predict 68 or 3D landmarks and is billed as "the world's most accurate face alignment network". MediaPipe's Face Mesh is another option, predicting **468 3D landmarks in real time** on-device [2]; it's fast and lightweight, though slightly less precise than heavy CNN models. All these tools are fully offline and GPU-accelerated. The face-alignment model (BSD-3 license) and MediaPipe (Apache 2.0) are open-source. The GTX 4090 can easily handle these models – RetinaFace (with ~27 million parameters) and FAN models will use only a few hundred MB of VRAM, so memory is ample [3] [4].

- **Feature Extraction & Measurements:** With stable landmarks (e.g. eye corners, nose tip, jawline points), the system computes key facial metrics. This includes distances, ratios, and angles that plastic surgeons care about: e.g. interpupillary distance, facial thirds proportions, nose width to face width ratio, jaw symmetry (left vs. right), chin projection, nasolabial angle, cervicomental (chin-neck) angle, etc. Many classical proportions (often related to aesthetic "ideal" ratios) can be calculated directly. For instance, inner canthus distance vs. eye width, or symmetry scores (distance between left/right feature positions). Tools like **Dynaface** (Apache 2.0 license) demonstrate this approach – it automatically places ~97 landmarks and outputs a variety of symmetry and facial measurements. These measurements are interpretable and can flag deviations (e.g. if one eye is 5mm higher than the other, or if nose length is longer than the ideal lower-third-of-face). Initially, threshold criteria for "issue" classification can be based on known aesthetic guidelines or surgeon input.

- **Issue Classification:** Building on these features, the pipeline can classify cosmetic issues either through rule-based heuristics or machine learning. A simple approach is defining rules: e.g. “if facial midpoint to eye corner distance differs >X% between left and right, flag asymmetry.” Over time, these rules can be refined by surgeon feedback (adjusting thresholds or adding new measurements). A more advanced approach is training a **multi-label classifier** that takes the face (or the computed metrics) as input and outputs probabilities for issues (like “chin retrusion” or “nasal asymmetry”). Open datasets like **CelebA** (which label attributes such as “Big Nose”, “Pointy Nose”, “Big Lips”, etc.) can provide a starting point for training models to recognize such features. For example, a deep CNN (ResNet or EfficientNet) pre-trained on CelebA attributes could be fine-tuned to detect cosmetic-relevant traits. The model could learn subtle cues beyond geometric ratios (like curvature of the nose bridge indicating a dorsal hump). However, since no public dataset exactly matches “surgical issues,” an iterative training strategy with surgeon-labeled images is crucial. The good news is that with a GTX 4090, even large models (e.g. a ResNet50 with 25M parameters) can be fine-tuned quickly offline. All major frameworks (PyTorch, TensorFlow) are open-source (BSD/Apache licenses) and can be used locally.

- **Pros:** Interpretable and precise results (actual measurements in mm or degrees) that align with clinical assessment; leverages well-validated open models (dlib, MediaPipe, FAN) for high accuracy landmark localization; fully offline and customizable. The approach is **extensible** – new measurements or classifications can be added as needed (e.g. if surgeons want to assess eyelid ptosis, one can measure eyelid aperture from landmarks). It’s also relatively easy to incorporate **feedback**: if the system’s classification is wrong, one can adjust the decision logic or retrain the classifier on the corrected label. Similarly, if a landmark is misplaced, the surgeon can manually correct it and those corrections (plus perhaps augmented data) can be used to fine-tune the landmark model to be more robust over time. Tools like Dynaface even allow exporting all measurements to CSV for analysis – surgeons could review these and set new thresholds or track improvement as the model learns.

- **Cons:** This pipeline may require significant **initial tuning and data** for the classification aspect. While landmark detection is pre-trained and high-accuracy, the translation from measurements to a specific “diagnosis” (e.g. “needs rhinoplasty”) is non-trivial. Without a sizable dataset of faces labeled with cosmetic issues, the ML classifier’s accuracy might be limited initially. The rule-based approach can miss nuanced issues or give false positives if a patient’s features are outside average ranges but still acceptable. Another challenge is handling variations in head pose or expressions – extreme angles or smiles can throw off measurements. Ensuring the input images are standardized (frontal neutral expression, etc.) mitigates this. Lastly, whereas the model can pinpoint *that* something is asymmetrical, providing a precise **segment or visualization** of the issue area might require extra steps (like overlaying the facial midline or highlighting the larger eye). A face-part segmentation model (e.g. an open-source face parsing CNN) could be integrated if per-region masking is needed, but that adds complexity.

2. Direct Deep Learning Attribute Classification

This approach forgoes explicit measurements and lets a neural network learn to identify cosmetic issues directly from images. It is essentially an **end-to-end CNN classifier/regressor** that outputs aesthetic judgments. For example, a model could be trained to output a “symmetry score” or flags like “jaw

asymmetry = Yes/No" from the raw image. Modern face recognition backbones or aesthetic scoring models can be repurposed here.

- **Pre-trained Models:** Open-source models like **VGG-Face, FaceNet, or InsightFace** provide deep features that encapsulate facial structure. By training a final layer on top of such features, the network can predict specific attributes. Researchers have even tackled beauty prediction with deep learning (e.g. **SCUT-FBP5500** dataset for facial beauty) – open implementations like *BeautyPredict AI* attempt to predict attractiveness scores, which often correlate with symmetry and proportion. We can leverage these as a starting point. For instance, take a ResNet50 pre-trained on VGGFace2 (which has learned facial structure representations) and fine-tune it on a custom dataset labeled for "ideal vs. needs improvement" in various facial regions. Transfer learning on a GTX 4090 will be swift, and frameworks like PyTorch (BSD license) make this accessible.

- **Training Data & Feedback:** Initially, one might use proxy labels. CelebA's **"Big Nose" attribute** could serve as a rough label for noses that might benefit from reduction, "Heavy Eyebrows" might correlate with brow lift cases, etc. Over time, surgeon feedback can create a proprietary dataset: each time the surgeon corrects the AI (e.g. indicating the nose was misclassified), that image can be added to the training set with the correct label. The system can periodically retrain or use online learning to incorporate these corrections, gradually improving its **clinical relevance**. The closed-source tool *Aesthetics AI* reportedly performs a similar proportion analysis to tailor treatments [5] – an open variant could be achieved by continuously learning from expert inputs.

- **Pros:** Potentially captures subtle features that rule-based methods might miss. A deep model can learn complex nonlinear relationships (like how a combination of slightly large nose and receding chin together influence facial harmony). It simplifies the pipeline – one network can directly output multiple issue categories, rather than stringing together detection-measurement-classification. This approach could also yield a confidence score for each prediction (helpful to let surgeons focus on high-confidence flags). With enough data, this method might outperform fixed geometric criteria, especially for subjective aesthetic judgments.

- **Cons: Data dependency** is the biggest drawback. Unlike landmarks (for which abundant training data and pre-trained models exist), there is no large public dataset labeled for "cosmetic surgery targets." The model's accuracy will heavily depend on incremental feedback and possibly synthetic data (e.g. augmenting images to simulate certain conditions). Until a robust dataset is built, the model might give inconsistent results. Moreover, deep models are **black boxes** – a surgeon might want to know *why* the AI labeled something an issue. Explaining a neural network's prediction can be difficult (though one could highlight facial heatmaps via Grad-CAM to indicate important regions). Also, training a separate model for each specific issue (nose, chin, etc.) might be necessary to keep things interpretable, which complicates maintenance. In practice, this approach is best combined with the landmark-based method: e.g. use a deep model to double-check or refine what the geometric analysis found, thereby harnessing the strengths of both.

(Approaches 1 and 2 above are not mutually exclusive – they often work together in an implemented system. Many practical solutions use robust landmark detection as a foundation, then apply learned models or rules on top.)

3. 3D Photogrammetry and Morphometric Analysis

A more elaborate approach uses **3D reconstruction** of the patient's face from multiple 2D images, yielding a realistic 3D model for analysis. This method prioritizes accuracy of measurements by overcoming perspective distortions inherent in single images. Using **photogrammetry**, one can capture a full facial geometry and then compute precise distances and angles in 3D space (much like a CT scan, but using just photographs).

- **Image Capture:** Typically, a **multi-view camera setup** or a single camera taking many photos around the patient's face is required. For example, capturing ~8–16 images from various angles (front, 45° left/right, profile, up/down angles) provides coverage. In one open-source protocol, 16 simultaneous photos taken in 0.4 seconds were used – however, one can do it sequentially with a single DSLR or even a smartphone, instructing the patient to stay still. Good lighting and a neutral expression are important for quality reconstruction.

- **Reconstruction Pipeline:** Open-source photogrammetry tools like **OpenMVG + OpenMVS**, or the fully integrated **Blender** workflow with add-ons, can be used. **OrtogOnBlender** and **RhinOnBlender** are Blender add-ons developed by Cícero Moraes et al., designed specifically for surgical planning. These use algorithms to stitch the photos into a 3D mesh of the face (employing Structure-from-Motion for camera pose estimation and Multi-View Stereo for surface reconstruction). The process is compute-intensive but offline – a 4090 GPU can accelerate parts of OpenMVS's dense reconstruction (it supports CUDA). After processing, you obtain a detailed 3D model of the patient's face with real-scale dimensions (often textured for realism).

- **Landmarks & Measurements:** In the 3D model, precise **landmark points** can be identified – either manually by the surgeon or automatically. The open-source workflow often allows manual placement of standard points (nasion, pogonion, alar bases, gonion, etc.), but researchers have also automated this by projecting 2D landmark detections from photos onto the 3D mesh. Once landmarks are set, the system can compute a comprehensive set of measurements: linear distances (e.g. bizygomatic width), curvilinear lengths (e.g. along the nose bridge), angles (e.g. nasal dorsum angle, mandibular plane angle), and volumetric or surface metrics if needed. One study demonstrated incorporating custom measurements like the **cervicomental angle** easily in an open-source 3D analysis. Multiple measurements can be combined or compared to population norms automatically. The output is essentially a **digital facial analysis report**, analogous to what a surgeon does with calipers and protractors on a patient or 3D scan, but faster and more objective. Impressively, photogrammetric 3D models have been found *comparable in accuracy to CT scans*, with negligible human error introduced.

- **Pros:** This approach offers **maximum accuracy** and detail. By working in true 3D, errors from head tilt or camera perspective are eliminated – you can rotate the model to any viewpoint. Symmetry assessment is particularly robust: the face can be mirrored in 3D to quantify asymmetry in depth. For example, one can automatically compute a symmetry plane and measure deviations for each half of the face (techniques for 3D asymmetry indices are documented in recent AI research). The 3D model also enables **simulation**: surgeons can virtually modify the model (e.g. "shave down" a dorsal hump or advance the chin) and directly visualize potential outcomes. Open-source tools allow creating surgical guides or morphs – e.g. the RhinOnBlender add-on can generate a 3D-printed guide for rhinoplasty based on the planned adjustments. In short, this pipeline not only flags issues but

integrates with planning. It's fully offline; Blender (GPL license) and libraries like OpenMVG (MPL2) are open-source. A GTX 4090 and a strong CPU (for some single-threaded parts) can reconstruct a high-res face in minutes. The **surgeon's feedback** loop here is often natural: if an automatically placed landmark is off, the surgeon moves it in the Blender interface – over time, these corrections could train an AI to place them better initially. If the system's analysis misses a measurement, the surgeon can add a custom point/angle definition and the software will include it for all future cases (thanks to the flexibility of open frameworks).

- **Cons:** The main drawbacks are **complexity and setup cost**. Unlike the 2D methods (which only need a single photo and run in seconds), photogrammetry requires multiple images and careful calibration. Setting up a multi-camera rig or taking many photos can be time-consuming in a clinical workflow (though one could streamline it with a turntable or a fixed camera array). The processing pipeline is heavy – while a 4090 can help, parts of the reconstruction may still take several minutes, and there is a learning curve to using Blender add-ons. Another consideration is that fully **automating** this pipeline is challenging: often a human must verify or adjust the 3D model and landmarks. Thus, initial use might involve the surgeon or an assistant spending some time in the software per patient. However, given that accuracy is the top priority, this trade-off might be acceptable. Lastly, while the tools are open-source, they are under active development by a niche community – documentation can be sparse. The developer integrating this should be comfortable with Blender's Python API and possibly troubleshooting openMVG/MVS outputs. In terms of licenses: Blender's GPL means any custom code that directly links with it should be open-sourced if distributed, but using the tool internally in a practice doesn't pose an issue. The add-ons like OrtogOnBlender are free to use (they bundle the needed libraries).

4. Hybrid & Other Approaches

Some solutions combine elements of the above. For instance, **2.5D single-image 3D models** can be generated with deep networks (e.g. **3DDFA** or **DECA** models that predict a 3D face mesh from one photo) – giving some of the benefits of 3D analysis without multi-view capture. These can be integrated into the pipeline to get approximate depth measures from a single image. Another hybrid idea is using **face parsing segmentation** (open models that label each facial region – eyes, nose, lips, etc.) to isolate regions and apply targeted analysis on each. For example, one could segment the nose and compare its shape to a dataset of "ideal" noses via a neural network. Each of these variations has pros/cons (e.g. single-image 3D is faster but less accurate than true photogrammetry). In practice, a full solution might use multiple subsystems: one for symmetry, one for proportions, one for texture/skin issues, etc., orchestrated together.

Closed-Source Benchmarks: It's worth noting a few proprietary tools for context. Products like **Visia®** (Canfield) provide facial analysis focusing on skin and proportions for cosmetic consultations, and **Crisalix®** offers 3D morphing for surgery simulation. Newer startups (e.g. *Qoves AI* or *Aedit*) market AI-driven facial assessments that suggest treatments. For example, *DeepFace AI* (not to be confused with Meta's algorithm) claims to create customized surgical plans, and *Aesthetics AI* focuses on proportion analysis. These systems are often cloud-based and not open-source, but they underscore the feasibility of what we seek. Our emphasis, however, is on open solutions that a clinic can deploy offline.

In summary, **Approach 1 (2D landmarks + analysis)** and **Approach 3 (3D photogrammetry)** emerge as the **two best solutions** given the priorities. The 2D landmark pipeline is relatively easier to implement and

can improve with incremental data, while the 3D pipeline offers unparalleled measurement accuracy and a rich basis for surgical planning. Below, we provide step-by-step implementation guides for each.

Solution 1: Deep Landmark & Feature Analysis Pipeline (Step-by-Step)

Objective: Build an offline system that takes one or a few 2D photos (frontal face, and optionally profile) and outputs an analysis of cosmetic issues, based on landmark measurements and learned feature classification. This system will run on a local PC with an NVIDIA 4090. All components will be open-source or locally hosted.

Steps:

1. **Environment Setup:** Install Python (preferably 3.9+ for latest libraries) and set up a virtual environment. Install PyTorch or TensorFlow (for GPU, use `torch==<latest>` with CUDA 11/12 support, or `tensorflow-gpu`). Ensure NVIDIA drivers and CUDA are installed for the 4090. Also install OpenCV (`opencv-python`), which will be used for image I/O and potentially face detection backends.

2. **Face Detection Model:** Integrate a robust face detector to locate faces in input images. We recommend **RetinaFace** (open-source implementation from the InsightFace project). Model Source: InsightFace provides a pretrained RetinaFace ResNet50 model (Apache License 2.0) which can be downloaded from their GitHub. Alternatively, use OpenCV's DNN with a pretrained model or dlib's frontal face detector as a fallback.

3. Download the RetinaFace model (e.g. `retinaface-R50` .onnx or .pth from InsightFace).
4. Write a function to load the model and run face detection on the input image. This should return the face bounding box (and five-point landmarks if using RetinaFace's output).

5. **Hardware:** The detection model will use minimal GPU memory. Running on the 4090, inference for a single image is <50ms.

6. **Facial Landmark Detection:** Use a high-accuracy landmark model to get detailed facial keypoints. One option is the **face-alignment** PyTorch library by Adrian Bulat. It provides a simple API: e.g. `fa = FaceAlignment(LandmarksType._2D, device='cuda')` then `preds = fa.get_landmarks(image)`. This returns 68 landmarks in pixel coordinates. Under the hood it uses an HRNet/FAN model (pretrained on 300W-LP), which is included in the package. License: BSD 3-Clause.

7. Install it via pip (`pip install face-alignment`) and its dependencies (will pull torch, numpy, etc.).
8. Alternatively, use **Mediapipe Face Mesh** if you want 468 landmarks. For Mediapipe, install `mediapipe-python` and use its FaceMesh solution (license Apache 2.0). Keep in mind Mediapipe runs on CPU by default; however it's real-time even on CPU. With GPU acceleration (if enabled via OpenGL), it's extremely fast. Since our focus is accuracy and we have a strong GPU, the face-

alignment CNN might be preferable for precision. You can even run both: e.g. use face-alignment for primary points and FaceMesh to get dense contour points if needed.

9. Ensure the image is cropped to the face ROI (detector output) before landmark prediction for best results. The face-alignment library can accept a detected box or will do internal detection (it uses SFD by default, but we already have RetinaFace for possibly better detection).
10. Verify the output by overlaying the landmarks on the image (for development/debugging). All points should map to expected facial features. On a well-centered front photo, the average error of these models is only a few pixels.

11. **Hardware:** The landmark model is a ~100MB CNN; on the 4090 it will run in a few milliseconds per face.

12. **Feature Computation (Geometry):** With landmarks in hand, compute key measurements:

13. **Facial Symmetry:** Determine the facial midline. For simplicity, you can use the line through the midpoint of the eyes and midpoint of the lips as an approximate symmetry axis. Reflect one side's landmarks onto the other side and compute RMS error or distance for pairs (e.g. left vs right jaw angle, eye positions). Dynaface, for example, computes symmetry scores per feature (eyes, eyebrows, mouth, etc.) [6] . You can implement similar: e.g. distance between left and right eye heights divided by a reference (like inter-eye distance) to quantify vertical asymmetry. Output overall symmetry percentage (100% = perfect symmetry).
14. **Proportions & Ratios:** Calculate ratios such as: (Nose length) / (Chin–Nose distance), (Lip width) / (Face width), (Interocular distance) / (Face width), etc. Compare these to ideal values from literature (for instance, the "Rule of Thirds" suggests equal thirds for hairline-to-eyebrow, eyebrow-to-nose, nose-to-chin distances). Flag any major deviations (beyond, say, 1–2 standard deviations of average). If available, incorporate known aesthetic indices – e.g. the **Marlowe–Ricketts line** for profile analysis or the **golden ratio mask** for frontal proportions – though those might require profile images or additional points.
15. **Feature-specific metrics:** For each region:
    - *Eyes:* Check eye aperture and inclination (canthality). E.g., a drooping eyelid could be flagged if the visible iris percentage is below a threshold – but that might need a classifier or segmentation of the eye. Simpler: measure if one eye's opening (distance between upper and lower eyelid landmarks) is significantly smaller than the other, which might indicate ptosis.
    - *Nose:* Measure nose width (alae width) relative to interpupillary distance; measure nose length vs face height. A common guideline is the nose width should roughly equal the intercanthal distance. Also check nasal symmetry: is the columella deviated? If profile photo is provided, measure dorsal hump or nasolabial angle (requires profile landmarks for nose tip, subnasale, upper lip).
    - *Chin/Jaw:* Measure chin prominence. If profile is available, the pogonion (chin tip) to a reference line (e.g. vertical from glabella) can indicate retrusion or protrusion. With just a front image, one proxy is the jaw width vs upper face width, or the menton point's deviation from midline.
    - *Lips:* Measure width of mouth vs nose or vs face. Also lip thickness if visible (height of upper/lower lip). If very thin compared to some norm, that's a feature often targeted (fillers).
    - *Others:* You can include forehead height (if hairline visible), facial width/height ratio (sometimes an aesthetic factor), etc.

16. Structure the code to compute these metrics from the landmark coordinates. It might be useful to create a small function for each measurement, for clarity and reusability. Store results in a dictionary.

17. **Issue Classification (Rules):** Based on the above measurements, derive an initial set of "diagnoses." For each metric, decide if it's in a normal range. For example:

18. If left–right symmetry score < 90% on eyes or jaw, mark **"Facial asymmetry"** issue.
19. If nose width > interpupillary distance, mark **"Wide nose"** (could correspond to potential rhinoplasty interest).
20. If nose length >> chin length (lower third), mark **"Long nose"**.
21. If chin is behind expected position (from profile or from proportion like jaw width too broad relative to chin point), mark **"Receding chin"** or **"Weak jawline."**
22. If lip height is very thin (say upper lip height < 0.1 * mouth width), mark **"Thin lips."**
23. Use known clinical standards when possible. For instance, nasofacial angle, if profile available, ideal ~36°. Or nasolabial angle (ideal ~95-105° for females). Deviation beyond 10° could be flagged.

24. Each rule can be coded with a threshold. Initially, use literature or expert estimates. (Surgeon input is valuable here – perhaps set these thresholds in config so they can be tweaked easily.)

25. **Issue Classification (ML Model – optional but recommended):** Augment the rule-based system with a machine learning model that learns from data:

26. **Data preparation:** If you have any labeled data (even small), e.g. a set of patient photos with notes like "deviated septum" or "jaw asymmetry", compile them. Also, consider using *CelebA* attributes as surrogate labels: for each relevant attribute ("Big Nose", "Narrow Eyes", "Attractive" etc.), you can train a classifier. Download CelebA (200k celebrity images with 40 attributes, released for research). Many of those attributes correlate with cosmetic considerations.
27. **Model selection:** Use a pre-trained CNN like ResNet50 (available from torchvision, under BSD-like license) to extract features. Replace the top layer with new outputs for each attribute of interest (or each issue category). For example, have outputs: {asymmetry, big_nose, receding_chin, etc.} as binary labels. You might start with 5-10 broad categories.
28. **Training:** Fine-tune the model on whatever dataset you have. If using CelebA, it's as simple as loading the images and attribute labels and training for those labels. You will need to align/resize faces – use the landmarks to align eyes horizontally for consistency. Since accuracy is priority and not real-time, you can afford a heavier model or even an ensemble. Train on the 4090 (it can handle large batches; this is all offline). Evaluate performance on a validation set if possible.
29. **Integration:** Once trained, this model can take a face crop as input and output probabilities for each learned issue. You'd then combine this with the rule-based findings. For example, if rules flagged "wide nose" and the CNN also has high confidence on "Big Nose," that reinforces the finding. If the CNN flags something the rules missed (say it predicts "chin issue" with high confidence even if your simple measurements didn't trigger), you can include a note like "AI suggests chin retrusion – review recommended."
30. Model source: If not training from scratch, there are open-source pre-trained attribute models. One such is an unofficial **CelebA attribute classifier** (there are GitHub repos, often MIT license, that provide pretrained weights for attribute prediction). Ensure any model you use is under a permissible license for your use-case (research vs commercial).

31. Over time, retrain this model with **feedback data**: every time surgeons correct the system, add that case (with corrected label) to your training set and periodically retrain to incorporate the new knowledge.

32. **Results Aggregation:** Design the output format. A concise report could be generated, for example:

33. **Facial Symmetry:** 88% (Slight asymmetry noted – right jaw is lower than left).
34. **Proportions:** Face length/width ratio = 1.35 (within normal range); Midface proportion (Nose length vs Chin) = High (recommend evaluation of chin projection).
35. **Detected Issues:**
    - *Nasal asymmetry:* Yes – Nose appears deviated towards left by ~3mm.
    - *Nasal proportion:* Nose width is larger than ideal (46mm vs ideal ~36mm).
    - *Chin retrusion:* Probable – Chin is behind ideal position (AI confidence 0.85).
    - *Jaw asymmetry:* Yes – Mandibular angles not symmetric (angle difference ~5°).
    - *Others:* Thin lips (borderline), Brow ptosis (No significant issue).
36. Use the rule outputs and ML outputs to fill in this report. Keep language clear that these are **AI findings, not diagnoses**, for medico-legal safety. The output could be in text form or overlaid on images (e.g. draw lines/angles on the face image for visualization).

37. If desired, generate an **annotated image**: for instance, draw landmark points and perhaps highlight regions of concern (you could draw a dashed line tracing an ideal symmetric jaw vs. the patient's jaw). OpenCV drawing functions can do this. This visual feedback is helpful for surgeons to quickly see what the AI is indicating.

38. **User Interface:** Depending on how this will be used by a developer or end-user (surgeon), you might create a simple GUI. For example, a small Qt or Tkinter app where a user can load a photo(s) and then see the analysis. Given that the requirement is not real-time, a simple interface is fine. Ensure that the app does not require internet (all models local). If security is important (patient data), confirm that everything runs locally with no data leaving the machine.

39. **Feedback Loop for Improvement:** Plan how surgeons will provide corrections and how those will update the system:

40. For classification corrections: If the AI flags "needs rhinoplasty" but the surgeon disagrees, the surgeon could press a "Incorrect" button next to that suggestion. The system should log that feedback (perhaps in a JSON or database with the image ID and the corrected label). Similarly, if the AI missed something ("It didn't mention the chin but I think chin augmentation is needed"), the surgeon should be able to add that note. All these feedback instances become new training examples. Periodically (say after accumulating 50 new cases), a developer or an automated process can retrain the ML models with the updated labels. This could even be scheduled during off-hours on the 4090, as training might take a few minutes to an hour depending on data size.
41. For landmark/segmentation corrections: If a landmark was off (suppose the outer canthus was misplaced due to heavy makeup), and the surgeon drags it to the correct spot on the image, you can capture the adjusted coordinates. Over time, you could fine-tune the landmark model (if you have access to its training procedure) on such corrected data – this is more involved, as you would need to add these as ground truth in the training set for that model and retrain or at least do a few epochs of fine-tuning. However, given the landmark model is already very accurate, this may not be frequently

needed. An alternative is to incorporate a **small adjustment module**: e.g. a lightweight model that learns the bias in landmark positioning from your data and corrects them (this is an advanced step and may not be necessary unless a consistent bias is observed).

42. Maintain version control for the models and a log of changes. Since surgeons will be using this as a clinical tool, validate each new model's outputs to ensure no regressions (you don't want the system to suddenly mis-detect something it used to do well).

43. **Testing and Calibration:** Before deploying widely, test the pipeline on a diverse set of images (different ages, ethnicities, lighting conditions). Because accuracy is paramount, you want to catch scenarios where it might fail (e.g. extreme expressions, occlusions like glasses, etc.). You might incorporate checks like: if face confidence is low or landmarks had to be extrapolated, flag the result as less reliable. This can prevent mis-analysis on poor inputs. Calibrate thresholds for issue flags so that the system isn't overzealous – a conservative approach is to flag fewer issues initially, to avoid "overcalling" normal variation as problems, and gradually tune sensitivity based on what surgeons find useful.

Licenses & Sources Recap (Solution 1): All components used are open-source: - *Face Detection:* RetinaFace (InsightFace) – MIT License; or OpenCV (BSD license); or dlib (Boost license). - *Landmarks:* face-alignment (BSD-3); MediaPipe (Apache 2.0). - *Frameworks:* PyTorch (BSD-style), TensorFlow (Apache 2.0). - *Training Data:* CelebA (freely available for non-commercial research; for commercial use, check their terms), or custom data collected under consent. - *Your Code:* Will likely be proprietary to the clinic, but since it links against these libraries, you can keep it closed source if desired (none of the chosen licenses are viral, except if using GPL components like some dlib models – but the ones listed avoid GPL). If you were to incorporate this into Blender or use GPL code, then you'd have to open-source those specific parts.

Hardware Requirements: A single GTX 4090 (24GB VRAM) is more than sufficient. In fact, this pipeline would run on a much smaller GPU, but the 4090 allows faster training and the option to use bigger models. A multi-core CPU (8+ cores, 32GB+ RAM) is recommended for general processing and potential training tasks. Disk space: if storing many images or datasets (CelebA is ~2GB), ensure you have 50–100GB free for data and model files. No internet is needed once all models and data are downloaded. The entire analysis of one face image (inference through detection, landmarks, and rule evaluation) should take only a fraction of a second on this hardware, meaning the bottleneck is more about the user interpreting results than the computation.

Solution 2: 3D Photogrammetry Pipeline (Step-by-Step)

Objective: Build an offline system that reconstructs a 3D model of a patient's face from photographs and provides precise measurements and visualizations of cosmetic metrics. This solution focuses on accuracy and detail, leveraging the GTX 4090 for computation. We will use open-source tools like Blender with specialized add-ons and computer vision libraries.

Steps:

1. **Set Up 3D Software and Libraries:** Install **Blender** (an open-source 3D modeling software, GPL license). Blender is the core platform where the 3D reconstruction and analysis will take place. The latest Blender (>= v3.x) is recommended for performance. Next, obtain the Blender add-ons

OrtogOnBlender and **RhinOnBlender** (developed by Cicero Moraes et al.). These can be found on GitHub (e.g., `cogitas3d/OrtogOnBlender`). Download the add-on `.zip` and install it into Blender (Edit -> Preferences -> Add-ons -> Install from File). OrtogOnBlender includes photogrammetry functions (wrapping OpenMVG/OpenMVS) and surgical planning tools for orthognathic surgery, while RhinOnBlender focuses on rhinoplasty (with additional tooling for nasal guide creation). Ensure these add-ons are enabled in Blender.

2. **Photogrammetry Tools:** If not bundled with the add-on, install **OpenMVG** (for Structure-from-Motion) and **OpenMVS** (for Multi-View Stereo). OrtogOnBlender often comes with these packaged or offers a one-click download within the add-on UI (check the documentation or GitHub issues if any manual setup is needed). Both OpenMVG and OpenMVS are open-source (MPL2 and GPL3 respectively). They perform the heavy lifting of turning images into a sparse 3D point cloud (OpenMVG) and then into a dense mesh (OpenMVS). The 4090 can accelerate OpenMVS's densification if compiled with CUDA – verify that OpenMVS is built with CUDA support. You might need to compile from source if pre-built binaries lack GPU support. Alternatively, **COLMAP** is another open-source SfM/MVS solution (GPL) that could be used similarly, but OrtogOnBlender is designed to integrate OpenMVG/MVS.

3. **Image Acquisition Protocol:** Develop a standard procedure for capturing patient photos:

4. Ideally, use a **DSLR or high-quality mirrorless camera** for consistent images. Smartphone cameras can work (as noted in literature: a smartphone photogrammetry protocol can yield results comparable to 3D scanners), but ensure consistent settings (same focal length, no heavy distortion or beauty filters).
5. Capture a **minimum of 8 images**: front view, left 45°, right 45°, left profile (90°), right profile, and optionally upper and lower angled views (looking slightly up at the chin and down at the forehead). More images (12-16) can improve quality. The add-on's documentation provides guidance; for example, a protocol might involve the patient turning 360° while photos are taken every ~30°.
6. Use a neutral background and even lighting to avoid shadows on the face (which can affect reconstruction). Ask the patient to maintain a neutral expression (no smile) and a natural head position.
7. Calibrate your camera if possible (to correct lens distortion). OpenMVG can handle unknown camera intrinsics by optimizing them, but providing an initial calibration (focal length, distortion coefficients) improves accuracy.

8. Once images are taken, transfer them to the workstation. For each patient, create a folder (e.g., `Patient123_photos/`) and put all the JPEGs there.

9. **3D Reconstruction Process:** Using Blender with OrtogOnBlender:

10. Open Blender and switch to the OrtogOnBlender interface (it might add a menu or panel in the 3D View or the Properties sidebar). There should be a section for Photogrammetry.
11. Load the patient images into the add-on. OrtogOnBlender likely provides a button like "Import Photos for 3D Reconstruction" or "Run Photogrammetry". Click that and select the folder of images.
12. Start the reconstruction. The add-on will internally call OpenMVG to compute features and matches between photos, then compute camera poses (this yields a sparse point cloud and camera calibration). Then it calls OpenMVS to compute a dense point cloud and mesh. This process can take

a few minutes depending on number of images and resolution. With a GTX 4090, dense reconstruction is significantly accelerated, but still expect on the order of 5–15 minutes for ~12 photos at 24MP each, for example.

13. Monitor the console for progress (Blender's system console will show OpenMVG/MVS logs). If any step fails (e.g. images not enough overlap, or an image was blurry and threw off camera alignment), you may need to recapture or adjust settings. Common pitfalls: if the patient moved significantly or the background had many features, the reconstruction might lock onto background. It helps to have the patient fill most of the frame and minimize background clutter.

14. Once completed, the add-on should import the resulting **3D face mesh** into Blender's scene. You'll likely see a textured model of the face. Save this Blender file for the patient (so you can revisit it later for planning or re-measuring if needed).

15. **Landmark Placement on 3D Model:** With the 3D face model ready, the next step is identifying key anatomical landmarks on it for measurements. There are two ways:

16. **Manual landmarking:** The add-ons provide a set of predefined points you can place. For example, RhinOnBlender might have slots for "Pronasale (nose tip)", "Subnasale", "Pogonion (chin tip)", "Tragion (ears)", etc. Activate a mode where you can click on the 3D model to mark these points. Because accuracy matters, a surgeon or trained assistant should do this – they can rotate the model to ensure they place the point exactly on the intended spot (e.g., the most projecting point of the chin for pogonion). OrtogOnBlender was used for orthognathic surgery planning, so it likely has points like orbitale, nasion, gnathion, etc. Use as many landmarks as needed for a full analysis (e.g., at least: bilateral exocanthion (eye corners), endocanthion, alar base points, subnasale, tip, supratip, pogonion, menton (chin bottom), gonion (jaw corners), zygion (cheek width), tragus or ear reference if needed).
17. **Automatic landmarking (experimental):** If you want to automate, you could project 2D landmarks onto the model. Since in Step 1 solution we already can get 2D landmarks on each photo, one approach is: take the 2D landmarks from the front photo (for example) and find their corresponding 3D coordinates. OpenMVG produces a sparse point cloud with feature correspondences – if some landmarks (like eye corners) were detected in the images, you could intersect rays to approximate their 3D location. However, doing this robustly is complex. Another approach is to use a 3D morphable model fitting (like 3DDFA) directly to get a full set of 3D points. Given the scope, it's acceptable to rely on manual marking, which surgeons may actually prefer for now (to trust the exact location).

18. After landmark placement, record their coordinates. The add-on might automatically label and save them. If not, you can use Blender's Python console to retrieve coordinates of specific vertices or empties that you placed as markers.

19. **Measurements & Analysis in 3D:** Now calculate the desired measurements using the 3D coordinates of those landmarks. This can be done via Blender's Python scripting or via the add-on's built-in functions:

20. **Linear distances:** Compute distances between key points: e.g., *Bi-zygoma width* (distance between left and right zygion points), *Jaw width* (distance between left and right gonion), *Face height* (nasion to menton), *Nose length* (nasion to subnasale or bridge length), *Nose width* (alar base distance), *Eye distance* (left endocanthion to right endocanthion). Because this is true 3D, these distances are in

millimeters (assuming the reconstruction is properly scaled). Often photogrammetry yields a scaled model, but sometimes scale is ambiguous. If the add-on allows, set a known distance (like a reference object or the distance between pupils as ~65 mm) to scale the model appropriately.

21. **Angles:** Compute angles between lines or planes: e.g., *Nasofrontal angle* (angle at nasion formed by glabella–nasion–nasal tip lines), *Nasolabial angle* (between columella and upper lip, using subnasale as vertex), *Mandibular plane angle* (between a line through gonion–gonion and a horizontal reference), *Intercanthal tilt* (if eyes are not level, small angle formed by line connecting canthi vs horizontal). Use vector math on coordinates to get these. In Blender's Python, you can define vectors and use dot products for angles.
22. **Symmetry analysis:** Determine the mid-sagittal plane of the face. One way: use the landmarks on the midline (nasion, subnasale, pogonion, menton) to define a plane. Reflect the coordinates of left-side points onto the right side and measure the distances to actual right-side points. Or compute the distance of each landmark from the mid-plane. This yields a quantitative asymmetry for each feature. The add-on or a script can automate this: e.g., output "Left cheek is 4mm fuller than right cheek" or "Chin deviated 2mm to left of center". Photogrammetry shines here because it captures depth differences (e.g. one cheekbone forward vs the other).
23. **Volume/Surface measurements (optional):** For certain issues like hemifacial microsomia or soft-tissue volume loss, volume could be relevant. One can mark a region (say one cheek) and calculate its surface area or volume vs the other side. This is advanced and likely beyond initial requirements, but know that with a 3D mesh such calculations are possible (Blender can compute volume of a mesh region).

24. It might be helpful to use an **automated report generator**. For example, OrtogOnBlender might output a text summary of common measurements once you place the standard landmarks. Check if the documentation mentions generating a "facial analysis report". If not, writing a custom Python script to do it is feasible. (The abstract from WJPS 2024 indicates they *automated combinations of measurements for comprehensive analysis* and generated a detailed report – likely through scripting Blender and perhaps using spreadsheets.)

25. **Results Presentation:** Present the findings to the user (surgeon) in a clear format:

26. Within Blender, you could create annotation text objects or use the add-on's interface to list measurements. However, expecting a surgeon to read Blender's UI might be less ideal. Instead, consider exporting the results.
27. **Generate a PDF or print-out:** The system can populate a template report with the numbers and perhaps simple graphics. For example, include a front and profile screenshot of the 3D model with landmarks indicated (Blender can be scripted to take snapshots from set angles). Draw lines or angles on these images to illustrate measurements (maybe use Blender's grease pencil or after export, use OpenCV to draw over the images). Then compile the images and text into a PDF. Python libraries like ReportLab or even just LaTeX (if familiar) could do this.
28. The report might have sections like: "Craniofacial Measurements vs Norms" – listing each measure, the patient's value, and normal range for that ethnicity/sex/age if available. Highlight any out-of-range values. For instance: "Nasal Width: 45mm (norm ~35mm for face width 140mm) – **High**" or "Chin deviation: 3.5mm left – **Noticeable asymmetry**".
29. Also present asymmetry visually: perhaps produce a *mirrored image comparison*. A classic approach is to mirror each half of the 3D face to show how the patient would look if perfectly symmetric to one side. Qoves and other aesthetic analysis sometimes do this to highlight differences. You can do this in Blender by duplicating the mesh, cutting it in half, mirroring one half, and overlaying – then

exporting an image. This is more for demonstration than measurement, but it can be compelling in consultations.

30. If rhinoplasty is a focus: include specific nasal analysis – e.g., dorsal profile line smoothness, any deviations, etc. The RhinOnBlender add-on can create a *surgical guide* if needed (it was used to design cutting guides for nasal bone adjustments). If the surgeon is interested, the guide (an STL model) can be generated and exported for 3D printing, though that's beyond analysis into planning/execution.

31. **Interactive Exploration:** One advantage of an offline local setup is that the surgeon can directly manipulate the model. Encourage usage of Blender's interactive tools:

32. The surgeon can rotate and zoom the 3D face to inspect features from any angle, something not possible with 2D photos alone.
33. They can **simulate changes**: For example, Blender's sculpting tools allow you to subtly modify the mesh (push the nose bridge down, augment the chin forward, etc.). This can act as a quick "simulation" of surgery outcomes. The add-ons might have specific tools for this (like moving jaw segments in orthognathic planning).
34. These simulations are not fully automated, but with a bit of Blender skill, it's doable. It can help verify if the identified issues, when corrected, indeed lead to an improved proportion/symmetry, reinforcing the AI's suggestions.

35. Since all data stays on the machine, patient confidentiality is preserved (no cloud processing), which is crucial for medical applications. Remind the team to follow privacy best practices (e.g., secure storage of these models and images, as they are patient-identifiable).

36. **Feedback & Improvement:** Incorporating feedback in this pipeline largely means refining the process and perhaps introducing more automation over time:

37. If a surgeon consistently finds that they have to add a certain measurement that our system didn't include (say, orbital hypertelorism distance), we can update the script to calculate that once the relevant landmarks are placed. Because the system is open-source and scriptable, it's straightforward to extend. For instance, adding a measurement of eye-to-eye angle or forehead curvature is just a matter of geometry once the points are available.
38. Landmark placement feedback: if some points are repeatedly tedious to place, consider automating them. You could, for example, train a small model using the accumulated data of manually placed points on meshes. This is complex (since it requires correlating 2D image features to 3D surfaces), but a workaround could be to project the 3D model to 2D and run a 2D landmark detector, then snap those to the 3D surface. Perhaps a future improvement is integrating something like **Statistical Shape Models** or a Morphable Model that can fit to the photogrammetry output for a fully automatic landmarking. The pipeline is modular, so these can be introduced later without changing the overall flow.

39. If photogrammetry fails or is impractical for some cases (e.g., patient can't provide multiple photos), have a contingency: fall back to the 2D approach (Solution 1). In practice, you might integrate both solutions – e.g., do a quick 2D analysis from the front photo immediately (for instant feedback), and then do the full 3D analysis when the model is ready for high precision results.

40. **Hardware and Performance Considerations:** The GTX 4090 will primarily accelerate the dense reconstruction stage (OpenMVS). Ensure in Blender's settings or the add-on config that GPU is enabled for compute. Also, have ample RAM – reconstructing a dense mesh from many photos can use 16GB+ RAM easily. A fast NVMe SSD is helpful for storing the temporary files (MVS will write depth maps, etc.). Keep the system's thermals in check during reconstruction since it's intensive. For reproducibility, maintain consistent versions of the photogrammetry libraries – updates can change results slightly. If you find a setup that works reliably, document it (e.g., OpenMVG v2.x, OpenMVS v1.x, Blender 3.x).

Licenses & Sources Recap (Solution 2): - Blender – GNU GPL v3 (free to use, must open-source any Blender *plugins* you write if distributed, but using it internally is fine). - OrtogOnBlender & RhinOnBlender add-ons – open-source (they don't explicitly state license in the snippet, but being on GitHub and referenced academically implies free use; likely GPL aligned with Blender). These add-ons were described in academic work as free alternatives for surgical planning. - OpenMVG – MPL2 (permissive; you can use it in closed software if needed). - OpenMVS – GPL3 (if you distribute software that links to it, you'd need to make that portion GPL; however, using it as an external program from Blender is fine). Since we're using it through Blender add-ons, Blender's GPL already covers it. - All these components are offline and open. The only caveat is if one wanted to integrate a proprietary photogrammetry engine (like Agisoft Metashape or RealityCapture) for possibly faster reconstruction – we won't, because open-source is preferred here. The chosen pipeline, while perhaps slightly less user-friendly than commercial software, has the advantage of cost-free use and **extensibility** (we can script and modify it at will).

Expected Hardware Requirements: - **GPU:** NVIDIA RTX 4090 – used for accelerating OpenMVS and possibly for viewport rendering in Blender. The 4090's 24GB VRAM can handle very high-resolution reconstructions (you could reconstruct millions of points into a mesh). - **CPU:** A multi-core CPU is heavily used by OpenMVG (feature extraction/matching can be multi-threaded) and parts of OpenMVS. A high clock speed helps for single-thread tasks too. A modern 12-core+ CPU (e.g., Ryzen 9 or Intel i9) is recommended. - **RAM:** 32GB minimum. If processing very high-res images or a large number of images, 64GB is safer to avoid swapping. - **Storage:** Each project (set of photos + reconstruction) can take a few hundred MB (the mesh and texture files). Also, OpenMVS will temporarily use disk for depth maps – ensure the temp directory has a few tens of GB free. An SSD is ideal for this scratch space. - **Peripherals:** A good camera for image capture (not exactly hardware *in* the system, but part of the solution). Optionally, a turntable or tripod setup to facilitate taking consistent photos around the patient. - **Time per analysis:** Plan on ~10 minutes for reconstruction plus a few minutes for landmarking and computation. This is longer than the 2D approach, but still possibly within a consultation session. Improvements like GPU-accelerated SfM (or reducing image count) can shorten it if needed, but at slight cost to detail.

By implementing these two solutions, a clinic would have a powerful toolkit: the 2D pipeline for quick screening and continuous AI learning, and the 3D pipeline for in-depth, quantitatively accurate analysis and surgical planning. Both run entirely offline on the given hardware, ensuring patient privacy and system responsiveness. Over time, with surgeon corrections fed back into the system, the accuracy and relevance of the analyses will only improve, ultimately providing a decision support system that augments the surgeon's expertise with data-driven insights.

Sources:

- Bulat, A. & Tzimiropoulos, G. (2017). *How far are we from solving the 2D & 3D Face Alignment problem?* (Face Alignment Network) – Implementation by 1adrianb (BSD License).
- Heaton, J. (2025). *Dynaface: Facial Symmetry Measurement Tool* (Apache-2.0) – 97-landmark detection and symmetry analysis.
- Stailey-Young, A. (2023). *RetinaFace* noted as most accurate open-source face detector [1].
- MediaPipe Face Mesh by Google – 468 landmarks in real-time on-device [2].
- Peixoto, D. *et al.* (2024). *Open Source Framework for Precise Digital Facial Analysis.* – Photogrammetry with 16 images, Blender reconstruction, automated measurements.
- Sobral, D. *et al.* (2021). *RhinOnBlender Add-on for Rhinoplasty Planning.* – Blender add-on for photogrammetry and surgical guide creation.
- Adel, M. *et al.* (2025). *AI-based 3D Facial Asymmetry Assessment.* – Automated landmark-based asymmetry index comparable to manual, demonstrating AI precision on 3D images.
- Specialist Practice Excellence (2023). *AI Tools for Plastic Surgeons.* – Mentions of BeautyPredict (symmetry-based outcome predictor) and Aesthetics AI (proportion analysis) [5] as context for what's being emulated with open-source tools.

---

[1] [3] [4] What's the Best Face Detector?. Comparing Dlib, OpenCV, MTCNN, and... | by Amos Stailey-Young | Python's Gurus | Medium
https://medium.com/pythons-gurus/what-is-the-best-face-detector-ab650d8c1225

[2] The 3D facial surface of 468 points in MediaPipe Face Mesh
https://www.researchgate.net/figure/The-3D-facial-surface-of-468-points-in-MediaPipe-Face-Mesh_fig4_367301766

[5] Best AI Tools for Plastic Surgeons & Plastics Practices | SPE
https://specialistpracticeexcellence.com.au/blog/best-ai-tools-agents-and-bots-for-plastic-surgeons-practices/

[6] GauravSharma0560/face-symmetry - GitHub
https://github.com/GauravSharma0560/face-symmetry