# 🛠️ How to Add Custom Gestures

The project is architected with the **Strategy Pattern** to make adding new gestures and effects plug-and-play. You can train the system to recognize a new hand sign in about 2 minutes.

## 1. Define the Gesture

Open `src/camera_input.py` and add your new gesture to the `CLASS_NAMES` configuration.

- **Tuple Format:** `("Name_of_Gesture", Number_of_Hands)`

```
CLASS_NAMES = [
    ("ThumbsUp", 1),
    ("Peace", 1),
    ("Gojo_Void", 1),
    # ... existing items
    ("Fireball_Jutsu", 2),  # <--- Add your new entry here
    ("Unknown", 1)
]
```

## 2. Record Training Data (Built-in Tool)

The app has an integrated data collection engine. No external tools needed.

1. Run the application: `python src/camera_input.py`.
2. Press **TAB** to cycle through the labels until you see your new gesture (`Fireball_Jutsu`) in the top-left UI.
3. Perform the gesture in front of the camera.
4. **Hold 'C'** to record data.
    - *Pro Tip:* Move your hands slightly (change angle, distance, and position) while recording to make the model robust against variation.
    - If your gesture is defined with 2 hands, the system will record automatically when it detects 2 hands in the webcam to collect training data.
    - *Goal:* Capture roughly **50-100 frames** (the counter in the console will tell you).

## 3. Retrain the Model

The ML pipeline is automated in a Jupyter Notebook.

1. Open `notebooks/train_model.ipynb`.
2. **Run All Cells.**
    - The notebook automatically loads `keypoints.csv` (which now contains your new data).
    - It retrains the Support Vector Machine (SVM).
    - It saves the updated model to `gesture_model.pkl`.
3. **Restart the App.** Your new gesture is now live!

## 4. (Optional) Add Visual Flair

Want your gesture to trigger a custom effect? Thanks to the **Strategy Pattern**, you just need to implement a standard interface.

Create a new class in `src/camera_input.py` that inherits from `VisualEffect`:

```
class FireballEffect(VisualEffect):
    def render(self, frame):
        if not self.is_active: return frame

        # Your custom OpenCV drawing logic (e.g., draw a circle)
        cv2.circle(frame, (400, 300), 50, (0, 0, 255), -1)
        return frame
```

Then, register it in the `main()` function's `effect_registry`:

```
effect_registry = {
    "Gojo_Void": GojoEffect(duration=30),
    "Fireball_Jutsu": FireballEffect(duration=45) # <--- Link Label to Strategy
}
```