Python FastAPI Backend Deployment Guide

📋 Table of Contents

🎯 Pre-Deployment Checklist

✅ Files Verification

Before deploying, ensure these files exist in `/home/ubuntu/basketball_app/python-backend/`:

- [x] `app/main.py` - Main FastAPI application
- [x] `app/pose_detection.py` - MediaPipe pose detection logic
- [x] `app/skeleton_drawing.py` - Skeleton overlay rendering
- [x] `app/anatomical_skeleton.py` - AI skeleton generation
- [x] `app/models.py` - Pydantic data models
- [x] `requirements.txt` - Python dependencies
- [x] `Dockerfile` - Container configuration
- [x] `.env.example` - Environment variables template

📦 Dependencies Verification

Your `requirements.txt` includes:

```
fastapi==0.109.2
uvicorn[standard]==0.27.1
python-multipart==0.0.9
mediapipe==0.10.9
opencv-python==4.9.0.80
Pillow==10.2.0
numpy==1.26.4
replicate==0.25.1
requests==2.31.0
pydantic==2.6.1
python-dotenv==1.0.1
starlette==0.36.3
```

Key Dependencies:
- **MediaPipe**: For real-time pose detection
- **OpenCV**: Image processing
- **Replicate**: AI-powered anatomical skeleton generation
- **FastAPI + Uvicorn**: Web framework and ASGI server

🔑 Required Environment Variables

All platforms require these environment variables:

Variable	Required	Default	Description
`REPLICATE_API_TOKEN`	✅ Yes	-	API token for AI skeleton generation
`ALLOWED_ORIGINS`	✅ Yes	-	Comma-separated list of allowed CORS origins
`HOST`	⚠️ Platform-specific	`0.0.0.0`	Server host (usually `0.0.0.0`)
`PORT`	⚠️ Platform-specific	`8000`	Server port (Railway uses dynamic `$PORT`)
`MEDIAPIPE_MODEL_COMPLEXITY`	Optional	`2`	Pose detection accuracy (0-2, higher = more accurate)

🔐 Environment Variables Reference

1. REPLICATE_API_TOKEN

Value: `r8_XVbSqNpDmahHdfRWDjmivN2ZNPk3MUH2w1N4x`

Purpose: Authenticates with Replicate API for AI-powered anatomical skeleton generation.

Where it's used:
- `/ai-skeleton` endpoint for detailed anatomical overlays
- Processes video frames with Replicate's ML models

⚠️ **Security Note:** Never commit this token to version control!

2. ALLOWED_ORIGINS

Format: Comma-separated URLs (no spaces)

Example:

```
ALLOWED_ORIGINS=https://your-abacus-frontend.vercel.app,http://localhost:3000,http://localhost:3001
```

Purpose: Controls which domains can make requests to your API (CORS security).

⚠️ **Critical:** You MUST include your deployed Abacus AI frontend URL here!

Finding Your Frontend URL:
1. Your frontend is deployed on Abacus AI
2. The URL format is typically: `https://[app-name].abacus.ai` or similar
3. Check your Abacus AI deployment dashboard for the exact URL
4. Add this URL to `ALLOWED_ORIGINS`

Common Patterns:

```
# For development + production
ALLOWED_ORIGINS=https://basketball-app.abacus.ai,http://localhost:3000

# With wildcards (use cautiously)
ALLOWED_ORIGINS=https://*.abacus.ai,http://localhost:3000

# Multiple environments
ALLOWED_ORIGINS=https://basketball-prod.abacus.ai,https://basketball-sta-
ging.abacus.ai,http://localhost:3000
```

3. HOST

Value: `0.0.0.0`

Purpose: Binds the server to all network interfaces, allowing external connections.

Platform Notes:
- Railway: Use `0.0.0.0`
- Render: Use `0.0.0.0`
- Vercel: Not applicable (serverless)

4. PORT

Default: `8000`

Platform-Specific Values:
- **Railway:** Use `$PORT` (Railway provides this dynamically)
- **Render:** Use `10000` or `$PORT`
- **Vercel:** Not applicable (serverless)

Important: Railway automatically injects `$PORT`, so set it as `PORT=$PORT` or let Railway handle it.

5. MEDIAPIPE_MODEL_COMPLEXITY

Value: `0`, `1`, or `2`

Default: `2` (highest accuracy)

Purpose: Controls MediaPipe pose detection model complexity.

Value	Performance	Accuracy	Use Case
0	Fastest	Lower	Real-time, low-latency
1	Balanced	Medium	General use
2	Slower	Highest	Production, detailed analysis

Recommendation: Keep at `2` for production unless experiencing performance issues.

🚀 Platform-Specific Deployment

Railway Deployment (⭐ RECOMMENDED)

Why Railway?

- ✅ Best support for Python + Docker
- ✅ Automatic HTTPS
- ✅ Simple environment variable management
- ✅ Built-in CI/CD from GitHub
- ✅ Free tier available ($5 credit/month)
- ✅ Excellent MediaPipe/OpenCV support

Step-by-Step Instructions

1. Prepare Your Repository

Option A: Use Existing GitHub Repository

Your code is already in: `https://github.com/baller70/BasketballAnalysisAssessmentApp.git`

Ensure the backend is in a clear directory structure:

```
BasketballAnalysisAssessmentApp/
├── python-backend/
│   ├── app/
│   ├── Dockerfile
│   ├── requirements.txt
│   └── .env.example
```

Option B: Create a New Repository (Backend Only)

```
cd /home/ubuntu/basketball_app/python-backend
git init
git add .
git commit -m "Initial backend commit"
git remote add origin https://github.com/yourusername/basketball-backend.git
git push -u origin main
```

2. Sign Up for Railway

1. Go to railway.app (https://railway.app)
2. Click **“Start a New Project”**
3. Sign in with GitHub (recommended for auto-deployment)

Screenshot Description:
- Railway homepage with purple gradient background
- “Start a New Project” button in center
- GitHub, GitLab, and email sign-in options

3. Create New Project

1. Click **“Deploy from GitHub repo”**
2. Authorize Railway to access your repositories
3. Select `BasketballAnalysisAssessmentApp` repository
4. Railway will detect the Dockerfile automatically

Screenshot Description:
- Repository selection screen
- List of your GitHub repositories
- Search bar at top
- “Deploy” button next to each repo

4. Configure Root Directory (Important!)

Since your backend is in a subdirectory:

1. Click on the deployed service
2. Go to **“Settings”** tab
3. Find **“Root Directory”** setting
4. Set it to: `python-backend`
5. Click **“Save”**

Screenshot Description:
- Settings page with tabs (General, Variables, Networking, etc.)
- Root Directory field highlighted
- Text input showing “python-backend”
- Blue “Save” button

5. Configure Environment Variables

1. Click **“Variables”** tab
2. Click **”+ New Variable”**
3. Add each variable:

Variable 1: REPLICATE_API_TOKEN

```
Name: REPLICATE_API_TOKEN
Value: r8_XVbSqNpDmahHdfRWDjmivN2ZNPk3MUH2w1N4x
```

Variable 2: ALLOWED_ORIGINS

```
Name: ALLOWED_ORIGINS
Value: https://your-abacus-frontend-url.com,http://localhost:3000
```

⚠️ **Replace `your-abacus-frontend-url.com` with your actual Abacus AI frontend URL!**

Variable 3: HOST

```
Name: HOST
Value: 0.0.0.0
```

Variable 4: PORT

```
Name: PORT
Value: $PORT
```

⚠️ **Important:** Railway automatically provides `$PORT`, this ensures your app uses it.

Variable 5: MEDIAPIPE_MODEL_COMPLEXITY

```
Name: MEDIAPIPE_MODEL_COMPLEXITY
Value: 2
```

Screenshot Description:
- Variables page with list of environment variables
- Each variable shown as a row with Name and Value columns
- “+ New Variable” button at top
- Eye icon to show/hide sensitive values

6. Enable Public Networking

1. Go to **“Settings”** > **“Networking”**
2. Click **“Generate Domain”**
3. Railway will create a public URL like: `https://i.ytimg.com/vi/zX-_NftC1K0/maxresdefault.jpg`
4. **Copy this URL** - you’ll need it later!

Screenshot Description:
- Networking settings page
- “Generate Domain” button
- Generated domain shown in blue text
- Copy icon next to the domain

7. Deploy!

Railway automatically deploys when you:
- Push to GitHub (if connected)
- Change environment variables
- Click “Deploy” manually

Monitor deployment:
1. Go to **“Deployments”** tab
2. Click on the latest deployment
3. View real-time logs
4. Wait for **“Build successful”** and **“Deployment live”**

Screenshot Description:
- Deployments page showing build logs
- Green “Deployment live” status
- Logs showing uvicorn server starting
- Timestamp for each log entry

8. Verify Deployment

Test the `/health` endpoint:

```
curl https://your-railway-url.up.railway.app/health
```

Expected Response:

```
{
  "status": "healthy",
  "version": "1.0.0",
  "mediapipe_available": true
}
```

Railway Troubleshooting

Problem: Build fails with “No Dockerfile found”
- Solution: Verify “Root Directory” is set to `python-backend`

Problem: Port binding error
- Solution: Ensure `PORT=$PORT` in environment variables

Problem: MediaPipe import errors
- Solution: Railway’s Docker support handles this automatically via your Dockerfile

---

Render Deployment

Why Render?

- ✅ Free tier available
- ✅ Good Docker support
- ✅ Automatic HTTPS
- ⚠️ Slower cold starts on free tier
- ⚠️ Free tier sleeps after inactivity

Step-by-Step Instructions

1. Sign Up for Render

1. Go to render.com (https://render.com)
2. Click **“Get Started”**

3. Sign up with GitHub (recommended)

Screenshot Description:
- Render homepage with blue/purple gradient
- “Get Started” button in navigation
- Signup options including GitHub

2. Create New Web Service

1. Click **“New +”** button (top right)
2. Select **“Web Service”**
3. Connect your GitHub repository
4. Select `BasketballAnalysisAssessmentApp`

Screenshot Description:
- Dashboard with “New +” dropdown menu
- Options: Web Service, Static Site, PostgreSQL, etc.
- Repository connection screen

3. Configure Service Settings

Basic Settings:
- **Name:** `basketball-backend`
- **Region:** Choose closest to your users
- **Branch:** `main`
- **Root Directory:** `python-backend`
- **Runtime:** `Docker`

Screenshot Description:
- Service configuration form
- Dropdown menus for region and runtime
- Text fields for name and root directory

4. Configure Build Settings

Build Command: (Leave empty - Docker handles this)

Start Command:

```
uvicorn app.main:app --host 0.0.0.0 --port $PORT
```

Render automatically injects `$PORT` environment variable.

Screenshot Description:
- Build settings section
- “Build Command” text area (empty)
- “Start Command” text area with uvicorn command

5. Choose Plan

- **Free:** $0/month (sleeps after 15min inactivity, 750 hours/month)
- **Starter:** $7/month (always on, better performance)

Recommendation: Start with Free, upgrade if needed.

Screenshot Description:
- Pricing tiers displayed as cards

- Free tier highlighted
- Feature comparison table

6. Add Environment Variables

Before clicking “Create Web Service”:

1. Scroll to **“Environment Variables”** section
2. Click **“Add Environment Variable”**
3. Add each variable:

Variables to Add:

```
REPLICATE_API_TOKEN=r8_XVbSqNpDmahHdfRWDjmivN2ZNPk3MUH2w1N4x
ALLOWED_ORIGINS=https://your-abacus-frontend-url.com,http://localhost:3000
HOST=0.0.0.0
PORT=10000
MEDIAPIPE_MODEL_COMPLEXITY=2
```

⚠️ **Replace `your-abacus-frontend-url.com` with your actual frontend URL!**

Screenshot Description:
- Environment variables section
- Key-value pairs for each variable
- “Add Environment Variable” button
- Eye icons to show/hide values

7. Deploy

1. Click **“Create Web Service”**
2. Render will:
   - Clone your repository
   - Build Docker image
   - Deploy container
   - Assign public URL

Deployment URL: `https://basketball-backend.onrender.com` (or similar)

Screenshot Description:
- Deployment in progress screen
- Real-time build logs
- Progress bar showing build stages
- Green “Live” badge when complete

8. Update CORS (If Needed)

After deployment, if you get CORS errors:

1. Go to **“Environment”** tab
2. Edit `ALLOWED_ORIGINS`
3. Add your Render URL: `https://basketball-backend.onrender.com`
4. Click **“Save Changes”** (triggers redeployment)

9. Verify Deployment

```
curl https://basketball-backend.onrender.com/health
```

Expected Response:

```
{
  "status": "healthy",
  "version": "1.0.0",
  "mediapipe_available": true
}
```

Render Troubleshooting

Problem: “Application failed to respond” error
- **Solution 1:** Check logs for port binding issues
- **Solution 2:** Ensure `PORT=10000` or `PORT=$PORT`
- **Solution 3:** Verify Dockerfile exposes correct port

Problem: Build fails with OpenCV errors
- **Solution:** Your Dockerfile already includes required system dependencies:

```
dockerfile
  RUN apt-get update && apt-get install -y \
      libgl1-mesa-glx \
      libglib2.0-0 \
      libsm6 \
      libxext6 \
      libxrender-dev
```

Problem: Free tier sleeps too often
- **Solution 1:** Upgrade to paid tier ($7/month)
- **Solution 2:** Use a keep-alive service (e.g., UptimeRobot) to ping every 14 minutes
- **Solution 3:** Accept 30-60 second cold start delays

Vercel Deployment

⚠️ Important Limitations

Vercel is **primarily designed for Node.js serverless functions**. Python support exists but has significant limitations:

Limitations:
- ❌ No native Docker support
- ❌ Serverless functions have 50MB size limit
- ❌ MediaPipe + OpenCV exceed size limits
- ❌ 10-second execution timeout (free tier)
- ❌ Limited Python package support

Verdict: ❌ **NOT RECOMMENDED** for this backend.

Alternative: Vercel for Frontend + Railway/Render for Backend

Recommended Architecture:
1. Deploy **Next.js frontend** on Vercel (optimal)
2. Deploy **Python backend** on Railway or Render
3. Configure CORS to allow Vercel frontend → Railway/Render backend

If You Must Try Vercel (Not Recommended)

Prerequisites

You would need to:
1. Convert FastAPI app to Vercel serverless functions
2. Remove MediaPipe/OpenCV (too large)
3. Use external APIs for pose detection
4. Rewrite significant portions of code

Steps (High-Level):

1. **Install Vercel CLI:**

```
npm install -g vercel
```

1. **Create `vercel.json`:**

```
{
  "builds": [
    {
      "src": "app/main.py",
      "use": "@vercel/python"
    }
  ],
  "routes": [
    {
      "src": "/(.*)",
      "dest": "app/main.py"
    }
  ]
}
```

1. **Strip Dependencies:**
   Remove from `requirements.txt`:
   - `mediapipe`
   - `opencv-python`
   - Use lighter alternatives or external APIs

2. **Deploy:**

```
vercel --prod
```

Why This Won't Work:
- Your backend relies heavily on MediaPipe and OpenCV
- These libraries are essential for pose detection
- No lightweight alternatives exist that maintain functionality

Conclusion: ❌ Stick with Railway or Render for this backend.

🔒 CORS Configuration

Understanding CORS

What is CORS?
Cross-Origin Resource Sharing (CORS) is a security mechanism that controls which domains can access your API.

Why It Matters:
Your frontend (Abacus AI) needs permission to call your backend API. Without proper CORS configuration, browsers will block requests.

Current CORS Setup

Your backend (`app/main.py`) includes:

```
app.add_middleware(
    CORSMiddleware,
    allow_origins=[
        "http://localhost:3000",
        "http://localhost:3001",
        "http://127.0.0.1:3000",
        "http://127.0.0.1:3001",
        "https://*.vercel.app",
    ],
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)
```

⚠️ You MUST Update ALLOWED_ORIGINS

Problem: Hardcoded origins in code won't work for your deployed Abacus AI frontend.

Solution: Use environment variable (already implemented).

In your deployment platform:

```
ALLOWED_ORIGINS=https://your-abacus-app.abacus.ai,http://localhost:3000
```

Dynamic CORS Configuration

Update your `app/main.py` (if needed):

```
import os

# Get allowed origins from environment variable
allowed_origins_str = os.getenv("ALLOWED_ORIGINS", "http://localhost:3000")
allowed_origins = [origin.strip() for origin in allowed_origins_str.split(",")]

app.add_middleware(
    CORSMiddleware,
    allow_origins=allowed_origins,
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)
```

Security Best Practices

✅ DO:

1. **Use specific domains:**
   ```
   bash
   ALLOWED_ORIGINS=https://basketball-app.abacus.ai
   ```
2. **List each domain explicitly:**
   ```
   bash
   ALLOWED_ORIGINS=https://prod.abacus.ai,https://staging.abacus.ai
   ```
3. **Include localhost for development:**
   ```
   bash
   ALLOWED_ORIGINS=https://prod.abacus.ai,http://localhost:3000
   ```

❌ DON'T:

1. **Avoid wildcard for production:**
   ```
   bash
   # Insecure!
   ALLOWED_ORIGINS=*
   ```
2. **Don't expose sensitive origins:**
   ```
   bash
   # Only include trusted domains
   ALLOWED_ORIGINS=https://your-app.com,https://malicious-site.com  # ❌
   ```

Testing CORS

Test from Browser Console:

1. Open your Abacus AI frontend in browser
2. Open Developer Tools (F12)
3. Go to Console tab
4. Run:

```
fetch('https://your-backend-url.up.railway.app/health')
  .then(res => res.json())
  .then(data => console.log(data))
  .catch(err => console.error('CORS Error:', err));
```

Expected Result (Success):

```
{
  status: "healthy",
  version: "1.0.0",
  mediapipe_available: true
}
```

CORS Error (Failure):

```
Access to fetch at 'https://...' from origin 'https://...' has been blocked by CORS
policy
```

Solution: Add your frontend domain to `ALLOWED_ORIGINS` and redeploy.

✅ Post-Deployment Steps

1. Copy Your Backend URL

Railway:
`https://basketball-backend-production-xxxx.up.railway.app`

Render:
`https://basketball-backend.onrender.com`

Copy this URL - you'll need it for the frontend!

2. Update Frontend Environment Variable

Location: Your Abacus AI frontend deployment settings

Variable to Update:

```
NEXT_PUBLIC_PYTHON_API_URL=https://your-backend-url.up.railway.app
```

Steps:
1. Go to your Abacus AI deployment dashboard
2. Find environment variables settings
3. Update `NEXT_PUBLIC_PYTHON_API_URL` with your backend URL
4. Redeploy the frontend

⚠️ **Important:** Remove trailing slash from URL!

✅ Correct:

```
NEXT_PUBLIC_PYTHON_API_URL=https://backend.railway.app
```

❌ Incorrect:

```
NEXT_PUBLIC_PYTHON_API_URL=https://backend.railway.app/
```

3. Update CORS in Backend

Ensure your frontend URL is in `ALLOWED_ORIGINS`:

```
ALLOWED_ORIGINS=https://basketball-app.abacus.ai,http://localhost:3000
```

On Railway/Render:
1. Go to Environment Variables
2. Edit `ALLOWED_ORIGINS`
3. Add your Abacus AI frontend URL
4. Save (triggers automatic redeployment)

4. Test All Endpoints

See Testing Your Deployment section below.

🧪 Testing Your Deployment

Test 1: Health Check

Purpose: Verify server is running and MediaPipe is available.

Command:

```
curl https://your-backend-url/health
```

Expected Response:

```
{
  "status": "healthy",
  "version": "1.0.0",
  "mediapipe_available": true
}
```

Troubleshooting:
- ❌ Connection refused → Check deployment status
- ❌ `mediapipe_available: false` → Check Dockerfile dependencies

Test 2: Analyze Endpoint (Image Upload)

Purpose: Test pose detection with a sample image.

Command:

```
curl -X POST https://your-backend-url/analyze \
  -F "file=@/path/to/basketball-image.jpg" \
  -H "Content-Type: multipart/form-data"
```

Expected Response:

```
{
  "keypoints": [
    {"x": 0.5, "y": 0.3, "z": -0.1, "visibility": 0.98, "name": "nose"},
    {"x": 0.48, "y": 0.28, "z": -0.05, "visibility": 0.95, "name": "left_eye"},
    ...
  ],
  "confidence": 0.92,
  "shooting_pose_detected": true,
  "image_dimensions": {"width": 1920, "height": 1080}
}
```

Troubleshooting:
- ❌ 400 Bad Request → Check file format (must be image)
- ❌ 500 Internal Server Error → Check logs for MediaPipe errors

Test 3: AI Skeleton Endpoint

Purpose: Test Replicate API integration.

Command:

```
curl -X POST https://your-backend-url/ai-skeleton \
  -F "file=@/path/to/basketball-image.jpg" \
  -F "strength=0.8" \
  -F "detail_level=high" \
  -H "Content-Type: multipart/form-data"
```

Expected Response:

```
{
  "output_image": "https://lh7-rt.googleusercontent.com/docsz/
AD_4nXfj5huwABX38xsU1SNq1yr-PUsWy-nVH5Cc6h0InAtdcvAHzXvmRDKAkPQy6JI1U9BD1M3-
foBI56GI_d1ASjzBlJQQ3SZJlMctBnCkzdx4QPCjN8Ry_x25znAL19pFriEsww1mVBQ?key=-ICjWANgPGN-
BY6zqswhs_w",
  "processing_time": 3.45
}
```

Troubleshooting:
- ❌ 401 Unauthorized → Check `REPLICATE_API_TOKEN`
- ❌ Timeout → Replicate API may be slow, increase timeout

Test 4: CORS Test (From Browser)

Purpose: Verify frontend can access backend.

Steps:
1. Open Abacus AI frontend in browser
2. Open Developer Tools (F12) → Console
3. Run:

```
fetch('https://your-backend-url/health')
  .then(res => res.json())
  .then(data => console.log('✅ CORS working:', data))
  .catch(err => console.error('❌ CORS error:', err));
```

Expected Result:

```
✅ CORS working: {status: "healthy", version: "1.0.0", ...}
```

If CORS Error:
1. Check `ALLOWED_ORIGINS` includes your frontend URL
2. Verify no typos in domain names
3. Ensure no trailing slashes
4. Redeploy backend after changes

Test 5: Full Integration Test

Purpose: Test complete workflow from frontend.

Steps:
1. Go to your Abacus AI frontend
2. Navigate to analysis page
3. Upload a basketball shooting image
4. Verify:
- ✅ Image uploads successfully
- ✅ Pose detection completes
- ✅ Skeleton overlay displays
- ✅ Elite shooter comparisons appear

Expected Behavior:
- Image preview shows immediately
- Loading spinner during analysis
- Skeleton overlay renders with keypoints
- No CORS errors in console

🔧 Troubleshooting Guide

Common Issues & Solutions

❌ CORS Errors

Error:

```
Access to fetch at 'https://backend.railway.app/analyze' from origin 'https://fron-
tend.abacus.ai'
has been blocked by CORS policy: No 'Access-Control-Allow-Origin' header is present
```

Cause: Backend doesn't allow requests from your frontend domain.

Solutions:

1. **Add frontend URL to ALLOWED_ORIGINS:**
   `bash`
   `ALLOWED_ORIGINS=https://frontend.abacus.ai,http://localhost:3000`

2. **Verify no typos:**
   - ✅ `https://frontend.abacus.ai` (correct)
   - ❌ `https://frontend.abacus.ai/` (trailing slash)
   - ❌ `http://frontend.abacus.ai` (http vs https)

3. **Check environment variable is loaded:**
   - View logs: Should see `allow_origins=[...]` on startup
   - Verify Railway/Render environment variables saved correctly

4. **Redeploy after changes:**
   - Environment variable changes require redeployment

Verification:

```
# Check CORS headers
curl -I https://your-backend-url/health \
  -H "Origin: https://your-frontend-url"

# Should include:
# Access-Control-Allow-Origin: https://your-frontend-url
```

❌ MediaPipe Import Error

Error:

```
ModuleNotFoundError: No module named 'mediapipe'
```

Cause: Dependencies not installed correctly.

Solutions:

1. **Verify requirements.txt includes mediapipe:**
   `txt`
   `mediapipe==0.10.9`
2. **Check Dockerfile installs system dependencies:**
   `dockerfile`
   `RUN apt-get update && apt-get install -y \`
   `libgl1-mesa-glx \`
   `libglib2.0-0 \`
   `libsm6 \`
   `libxext6 \`
   `libxrender-dev`
3. **Rebuild from scratch:**
   - Railway: Trigger manual redeploy
   - Render: Clear build cache and redeploy
4. **Check build logs:**
   - Look for `Successfully installed mediapipe-0.10.9`
   - Watch for “Failed to build” errors

❌ OpenCV Import Error

Error:

```
ImportError: libGL.so.1: cannot open shared object file
```

Cause: Missing OpenCV system dependencies.

Solution:

Your Dockerfile already includes the fix. If error persists:

1. **Verify Dockerfile includes:**
   `dockerfile`
   ```
   RUN apt-get update && apt-get install -y \
       libgl1-mesa-glx \
       libglib2.0-0
   ```
2. **Rebuild Docker image:**
   `bash`
   ```
   docker build --no-cache -t basketball-backend .
   ```
3. **Alternative: Use opencv-python-headless:**
   In `requirements.txt`:
   `txt`
   ```
   opencv-python-headless==4.9.0.80
   ```
   (This removes GUI dependencies)

❌ Port Binding Error

Error:

```
OSError: [Errno 98] Address already in use
```

Cause: Wrong PORT configuration.

Solutions:

Railway:

```
PORT=$PORT
```

Railway injects dynamic port.

Render:

```
PORT=10000
# or
PORT=$PORT
```

Local Testing:

```
PORT=8000
```

Verification:
Check logs for:

```
INFO:     Uvicorn running on http://0.0.0.0:8000 (Press CTRL+C to quit)
```

❌ Replicate API Errors

Error 1: 401 Unauthorized

```
{"detail": "Invalid Replicate API token"}
```

Solution:
- Verify `REPLICATE_API_TOKEN` is set correctly
- Check for extra spaces or newlines
- Ensure token starts with `r8_`

Error 2: 429 Rate Limited

```
{"detail": "Replicate API rate limit exceeded"}
```

Solution:
- Wait and retry
- Upgrade Replicate plan for higher limits
- Implement caching for repeated requests

Error 3: Timeout

```
replicate.exceptions.ReplicateError: Request timeout
```

Solution:
- Replicate's AI models can be slow (30+ seconds)
- Increase timeout in code:

```
python
  import replicate
  replicate.Client(api_token=REPLICATE_API_TOKEN, timeout=60)
```

❌ Image Upload Fails

Error:

```
{"detail": "File must be an image"}
```

Cause: Invalid file type or corrupted upload.

Solutions:

1. **Check file type:**
   - Supported: JPG, JPEG, PNG, WebP
   - Not supported: GIF, SVG, HEIC

2. **Verify Content-Type header:**

```
bash
   curl -X POST ... -F "file=@image.jpg" \
     -H "Content-Type: multipart/form-data"
```

3. **Check file size:**
   - Max recommended: 10 MB
   - Larger files may timeout

4. **Test with sample image:**
   ```bash
   # Download test image
   curl -o test.jpg https://dummyimage.com/svga

# Test upload
curl -X POST https://your-backend-url/analyze \
-F “file=@test.jpg”
```

❌ Frontend Can’t Connect

Symptoms:
- Network errors in browser console
- “Failed to fetch” errors
- Requests never reach backend

Solutions:

1. **Verify NEXT_PUBLIC_PYTHON_API_URL:**
   - Check Abacus AI frontend environment variables
   - Should be: `https://your-backend-url`
   - No trailing slash!

2. **Check backend is running:**
   ```bash
   curl https://your-backend-url/health
   ```

3. **Verify HTTPS:**
   - Backend URL must use HTTPS (not HTTP)
   - Railway and Render provide automatic HTTPS

4. **Check CORS (again):**
   - See CORS troubleshooting section above

5. **Test from command line:**
   ```bash
   # If this works but frontend doesn't, it's a CORS issue
   curl https://your-backend-url/health
   ```

❌ Deployment Succeeds but API Returns 404

Error:

```
{"detail": "Not Found"}
```

Cause: Incorrect root directory or routing.

Solutions:

1. **Verify endpoint paths:**
   - ✅ `https://backend-url/health`
   - ❌ `https://backend-url/api/health`
   - ❌ `https://backend-url/python-backend/health`

2. **Check Root Directory setting:**
   - Railway: Should be `python-backend`
   - Render: Should be `python-backend`

3. **Verify main.py location:**
   ```
   python-backend/
   └── app/
       └── main.py  ← Should be here
   ```

4. **Check start command:**
   ```bash
   uvicorn app.main:app --host 0.0.0.0 --port $PORT
   ```
   Not:
   ```bash
   uvicorn main:app  # ❌ Wrong
   ```

❌ Slow Response Times

Symptoms:
- Requests take 10+ seconds
- Timeouts on image analysis

Solutions:

1. **Reduce MediaPipe complexity:**
   ```bash
   MEDIAPIPE_MODEL_COMPLEXITY=1  # Instead of 2
   ```

2. **Optimize image size:**
   - Resize images before upload
   - Recommended: 1920x1080 or smaller

3. **Upgrade hosting plan:**
   - Railway: Hobby plan ($5/month)
   - Render: Starter plan ($7/month)
   - Free tiers have limited CPU/memory

4. **Add caching:**
   - Cache pose detection results
   - Avoid re-processing same images

5. **Use async endpoints:**
   - Already implemented with `async def`
   - Ensure Uvicorn uses multiple workers:
   ```bash
   uvicorn app.main:app --workers 4
   ```

❌ Memory Errors

Error:

```
MemoryError: Unable to allocate array
```

Cause: Large images or limited RAM.

Solutions:

1. **Resize images before processing:**
   Add to `pose_detection.py`:
   ```
   python
   max_dimension = 1920
   if image.shape[0] > max_dimension or image.shape[1] > max_dimension:
       scale = max_dimension / max(image.shape[:2])
       image = cv2.resize(image, None, fx=scale, fy=scale)
   ```

2. **Upgrade hosting plan:**
   - Free tiers: 512 MB RAM
   - Paid tiers: 2+ GB RAM

3. **Reduce model complexity:**
   ```
   bash
   MEDIAPIPE_MODEL_COMPLEXITY=0
   ```

Getting Help

If issues persist:

1. **Check deployment logs:**
   - Railway: Deployments tab → View Logs
   - Render: Logs tab (left sidebar)

2. **Enable debug mode:**
   Add to environment variables:
   ```
   bash
   LOG_LEVEL=DEBUG
   ```

3. **Test locally:**
   ```
   bash
   cd /home/ubuntu/basketball_app/python-backend
   pip install -r requirements.txt
   uvicorn app.main:app --reload
   ```

4. **Contact support:**
   - Railway: railway.app/help (https://railway.app/help)
   - Render: render.com/support (https://render.com/support)

📝 Deployment Checklist

Pre-Deployment

- [ ] All files present in `/home/ubuntu/basketball_app/python-backend/`
- [ ] `requirements.txt` includes all dependencies
- [ ] `Dockerfile` configured correctly
- [ ] `.env.example` documented
- [ ] Code pushed to GitHub

During Deployment

- [ ] Platform account created (Railway/Render)
- [ ] Repository connected
- [ ] Root directory set to `python-backend`
- [ ] Environment variables configured:
- [ ] `REPLICATE_API_TOKEN`
- [ ] `ALLOWED_ORIGINS` (with frontend URL)
- [ ] `HOST=0.0.0.0`
- [ ] `PORT` (platform-specific)
- [ ] `MEDIAPIPE_MODEL_COMPLEXITY=2`
- [ ] Public domain generated

Post-Deployment

- [ ] `/health` endpoint returns 200 OK
- [ ] `mediapipe_available: true` in health check
- [ ] Frontend environment variable updated:
- [ ] `NEXT_PUBLIC_PYTHON_API_URL` set to backend URL
- [ ] Frontend redeployed
- [ ] CORS test passes from browser
- [ ] `/analyze` endpoint tested with sample image
- [ ] `/ai-skeleton` endpoint tested
- [ ] Full integration test completed

Monitoring

- [ ] Set up uptime monitoring (optional)
- [ ] Configure error alerts
- [ ] Monitor API usage/costs

🎉 Success!

If all tests pass, your Python FastAPI backend is successfully deployed and connected to your Abacus AI frontend!

Next Steps:
1. Share the frontend URL with users
2. Monitor performance and errors
3. Scale as needed

4. Consider adding:
- Rate limiting
- API authentication
- Response caching
- Monitoring/analytics

Useful Links:
- Frontend (Abacus AI): `https://your-frontend-url`
- Backend API: `https://your-backend-url`
- API Documentation: `https://your-backend-url/docs` (FastAPI auto-generated)
- Backend Health: `https://your-backend-url/health`

📚 Additional Resources

- FastAPI Documentation (https://fastapi.tiangolo.com/)
- Railway Documentation (https://docs.railway.app/)
- Render Documentation (https://render.com/docs)
- MediaPipe Pose Guide (https://google.github.io/mediapipe/solutions/pose.html)
- Replicate API Docs (https://replicate.com/docs)

Document Version: 1.0.0
Last Updated: December 4, 2025
Maintainer: Basketball Analysis Team