PyInstaller Compilation Fixes - Summary

This document summarizes all the fixes applied to make the Human Blur Tool work correctly when compiled with PyInstaller.

Issues Fixed

1. ✅ Logo Not Appearing in Compiled App

Problem:
- The GEN logo (logo.png) was not visible in the compiled application window
- The code was using `Path(__file__).parent` which doesn't work in frozen executables

Root Cause:
- When PyInstaller freezes an application, it extracts files to a temporary directory
- The `__file__` variable points to the wrong location in frozen mode
- PyInstaller stores the actual resource path in `sys._MEIPASS`

Solution:
Added a `get_resource_path()` helper function in `gui_enhanced.py`:

```
def get_resource_path(relative_path):
    """
    Get absolute path to resource, works for both development and PyInstaller frozen
mode.
    """
    try:
        # PyInstaller creates a temp folder and stores path in _MEIPASS
        base_path = Path(sys._MEIPASS)
    except AttributeError:
        # Running in normal Python environment
        base_path = Path(__file__).parent

    return base_path / relative_path
```

Updated logo loading code to use this function:

```
logo_path = get_resource_path("logo.png")
```

Files Modified:
- `gui_enhanced.py` - Added resource path helper function and updated logo loading

---

2. ✅ Processing Hangs/Doesn't Work in Compiled Version

Problem:
- When clicking "Process Media" in the compiled app, it would show "processing" but never actually process
- The application would hang indefinitely
- No error messages were shown

Root Cause:
- PyTorch and ultralytics use multiprocessing internally
- Multiprocessing doesn't work correctly in frozen executables without proper initialization
- Missing `freeze_support()` call causes child processes to hang

Solutions Implemented:

A. Added multiprocessing.freeze_support()

Added to the main entry point in `gui_enhanced.py`:

```
def main():
    """Main entry point for enhanced GUI application."""
    # Fix for multiprocessing in frozen executables
    import multiprocessing
    multiprocessing.freeze_support()

    root = tk.Tk()
    # ... rest of the code
```

B. Created Runtime Hook

Created `runtime_hook.py` to initialize multiprocessing before the app starts:

```
import sys
import multiprocessing

# Fix for multiprocessing in frozen executables
if getattr(sys, 'frozen', False):
    multiprocessing.freeze_support()

    try:
        multiprocessing.set_start_method('spawn', force=True)
    except RuntimeError:
        pass
```

C. Added All Hidden Imports

Created comprehensive hidden imports list in `HumanBlurTool.spec`:

```
hiddenimports=[
    # GUI and image processing
    'PIL._tkinter_finder',
    'PIL.Image',
    'PIL.ImageTk',
    'tkinter',
    'tkinter.ttk',

    # OpenCV and image processing
    'cv2',
    'numpy',
    'numpy.core._multiarray_umath',

    # YOLO and ultralytics (critical for processing)
    'ultralytics',
    'ultralytics.models',
    'ultralytics.models.yolo',
    'ultralytics.nn',
    'ultralytics.nn.modules',
    'ultralytics.utils',
    'ultralytics.engine',
    'ultralytics.engine.results',
    'ultralytics.engine.predictor',

    # PyTorch (required by YOLO)
    'torch',
    'torch._C',
    'torch.nn',
    'torch.nn.functional',
    'torchvision',
    'torchvision.ops',

    # Other dependencies
    'yaml',
    'matplotlib',
    'scipy',
    'pandas',
    'tqdm',
]
```

Files Created/Modified:
- `gui_enhanced.py` - Added multiprocessing.freeze_support() in main()
- `runtime_hook.py` - NEW file for multiprocessing initialization
- `HumanBlurTool.spec` - NEW file with complete PyInstaller configuration

New Files Created

1. HumanBlurTool.spec

Purpose: Complete PyInstaller configuration file with all necessary settings

Key Features:
- Includes logo.png as bundled data
- Comprehensive hidden imports for all dependencies
- Runtime hook integration
- Optimized settings (UPX compression, single-file executable)

- macOS app bundle configuration with proper metadata
- Excludes unnecessary modules to reduce size

Usage:

```
pyinstaller HumanBlurTool.spec --clean
```

2. runtime_hook.py

Purpose: Initialize multiprocessing correctly in frozen executables

What it does:
- Detects if running in frozen mode
- Calls `multiprocessing.freeze_support()`
- Sets the multiprocessing start method to 'spawn'

Integration: Automatically included via `runtime_hooks` in the spec file

3. build.sh (macOS/Linux)

Purpose: Automated build script for Unix-like systems

Features:
- Checks for PyInstaller and installs if needed
- Cleans previous builds
- Builds using the spec file
- Displays clear success/failure messages
- Shows platform-specific instructions

Usage:

```
chmod +x build.sh
./build.sh
```

4. build.bat (Windows)

Purpose: Automated build script for Windows

Features:
- Same as build.sh but for Windows
- Handles Windows-specific paths and commands

Usage:

```
build.bat
```

Code Changes Summary

gui_enhanced.py

Changes made:

1. **Added resource path helper** (lines 19-36):
   ```
   python
      def get_resource_path(relative_path):
          # ... handles frozen and unfrozen modes
   ```
2. **Updated logo loading** (line 291):
   ```python
   # OLD:
   logo_path = Path(__file__).parent / "logo.png"

# NEW:
logo_path = get_resource_path("logo.png")
```

1. **Added multiprocessing support** (lines 909-911):
   ```
   python
      def main():
          import multiprocessing
          multiprocessing.freeze_support()
          # ...
   ```

No other code changes were needed - the rest of the application works as-is!

Testing the Fixes

Before Building

1. Ensure all dependencies are installed:
   ```
   bash
      pip install -r requirements.txt
      pip install pyinstaller
   ```
2. Verify the GUI works in development mode:
   ```
   bash
      python gui_enhanced.py
   ```

Building

Easy way:

```
# macOS/Linux:
./build.sh

# Windows:
build.bat
```

Manual way:

```
pyinstaller HumanBlurTool.spec --clean
```

Testing the Compiled App

1. **Test logo display:**
   - Open the compiled app
   - Verify the GEN logo appears in the top-right corner

2. **Test processing:**
   - Select a test image or video
   - Click “Process Media”
   - Verify it actually processes (progress bar moves, completion message appears)
   - Check the output file is created correctly

3. **Test batch processing:**
   - Select a folder with multiple files
   - Process the folder
   - Verify all files are processed correctly

Platform-Specific Testing

macOS:

```
open dist/HumanBlurTool.app
```

If you get “damaged app” warning:

```
xattr -cr dist/HumanBlurTool.app
open dist/HumanBlurTool.app
```

Windows:

```
dist\HumanBlurTool.exe
```

Linux:

```
chmod +x dist/HumanBlurTool
./dist/HumanBlurTool
```

Why These Fixes Work

Resource Path Fix

PyInstaller extracts bundled files to a temporary directory when running the executable. The `sys._MEIPASS` attribute contains the path to this temporary directory. Our helper function checks for this attribute and uses it when available, falling back to the normal path when running as a script.

Multiprocessing Fix

Python's multiprocessing module needs special handling in frozen executables because:

1. **Process spawning works differently:** In frozen mode, new processes can't just import the modules they need - they need to be told they're part of a frozen app
2. **freeze_support() is required:** This function tells multiprocessing how to handle frozen executables
3. **Hidden imports are critical:** PyInstaller needs to know about all the modules that will be imported dynamically at runtime

Without these fixes, child processes would try to start but couldn't properly initialize, causing hangs.

Hidden Imports

PyInstaller analyzes code to find imports, but it can't detect:
- Dynamic imports (using `__import__()` or `importlib`)
- Imports inside try/except blocks
- Plugins and extensions loaded at runtime

Ultralytics/YOLO uses many dynamic imports, so we explicitly tell PyInstaller about all necessary modules.

Debugging Tips

If issues persist after applying these fixes:

Enable Console Output

In `HumanBlurTool.spec`, change:

```
console=False,  # Change to True
```

This shows console output and error messages.

Check for Missing Imports

If you get "ModuleNotFoundError":
1. Note the missing module name
2. Add it to `hiddenimports` in the spec file
3. Rebuild

Verify Resource Bundling

Check if logo is included:

```
# macOS
unzip -l dist/HumanBlurTool.app/Contents/MacOS/HumanBlurTool | grep logo.png

# Linux/Windows (if using --onefile)
# The file is extracted at runtime to sys._MEIPASS
```

Test Multiprocessing

Add debug prints in `runtime_hook.py`:

```
if getattr(sys, 'frozen', False):
    print("Running in frozen mode")
    multiprocessing.freeze_support()
    print("Multiprocessing initialized")
```

File Size Considerations

Expected sizes:
- Windows: 400-800 MB
- macOS: 450-900 MB
- Linux: 400-800 MB

Why so large?
- PyTorch: ~200-400 MB
- OpenCV: ~50 MB
- Ultralytics + YOLO models: ~10-50 MB
- Python runtime: ~50 MB
- Other dependencies: ~100-200 MB

To reduce size:
1. Use CPU-only PyTorch (saves ~100-200 MB)
2. Use directory-based build instead of single file
3. Exclude unused modules (already done in spec file)

Distribution Checklist

Before distributing the compiled executable:

- [ ] Test on a clean machine without Python installed
- [ ] Verify logo displays correctly
- [ ] Test single file processing
- [ ] Test batch folder processing
- [ ] Test both image and video processing
- [ ] Verify all blur/mask modes work
- [ ] Check error messages are user-friendly
- [ ] Include README with system requirements
- [ ] For macOS: Code sign if possible
- [ ] For Windows: Consider creating installer (Inno Setup)
- [ ] Test on target OS version

Additional Notes

Model Downloads

YOLO models are NOT bundled by default. The first time a user runs the app, it will:
1. Connect to the internet
2. Download the selected model (~6MB for yolov8n-seg.pt)
3. Cache it in `~/.cache/ultralytics/`

For offline distribution, see the “YOLO models not downloading” section in BUILD_EXECUTABLE.md.

Audio Handling

The “Keep audio in output videos” feature requires ffmpeg. Users need to have ffmpeg installed separately, or you can bundle it with your executable.

Cross-Platform Compatibility

These fixes work on all platforms (Windows, macOS, Linux). The spec file automatically detects the platform and builds appropriately.

Support

If you encounter issues not covered here:

1. Check BUILD_EXECUTABLE.md for detailed troubleshooting
2. Enable console mode for debugging
3. Check PyInstaller logs in `build/HumanBlurTool/` directory
4. Contact: apps@globalemancipation.ngo

Last Updated: November 19, 2025
PyInstaller Version Tested: 6.x
Python Version: 3.8+