Building Executables for Human Blur Tool GUI

This guide explains how to build standalone executables for the Human Blur Tool GUI on Windows, macOS, and Linux.

✨ **NEW: We now include a pre-configured PyInstaller spec file and build scripts for easy compilation!**

Table of Contents

Quick Start

The Easy Way (Recommended) 🚀

We've created automated build scripts that handle everything for you!

Windows:

```
build.bat
```

macOS / Linux:

```
./build.sh
```

These scripts will:
- Check for PyInstaller and install if needed
- Clean previous builds
- Build using the optimized configuration
- Show you where to find your executable

Prerequisites

Before building executables, ensure you have:

1. **Python 3.8 or higher** installed

2. **All dependencies** installed:
   ```bash
   pip install -r requirements.txt
   ```
3. **PyInstaller** (the build scripts will install it automatically, or run):
   ```bash
   pip install pyinstaller
   ```

Building on Windows

Method 1: Using the Build Script (Easiest)

1. **Open Command Prompt or PowerShell** and navigate to the project directory:
   ```cmd
   cd path\to\human-blur-tool
   ```

2. **Run the build script**:
   ```cmd
   build.bat
   ```

3. **Find your executable**:
   - Location: `dist\HumanBlurTool.exe`

Method 2: Using PyInstaller with Spec File (Manual)

1. **Navigate to the project directory**:
   ```cmd
   cd path\to\human-blur-tool
   ```

2. **Build using the spec file**:
   ```cmd
   pyinstaller HumanBlurTool.spec --clean
   ```

3. **Find your executable**:
   - Location: `dist\HumanBlurTool.exe`

Testing the Windows Executable

```
cd dist
HumanBlurTool.exe
```

Optional: Create Installer with Inno Setup

For professional distribution:
- Download Inno Setup (https://jrsoftware.org/isinfo.php)
- Create a setup script to bundle the executable
- This creates a professional installer for Windows users

Building on macOS

Method 1: Using the Build Script (Easiest)

1. **Open Terminal** and navigate to the project directory:
   ```
   bash
   cd /path/to/human-blur-tool
   ```
2. **Run the build script**:
   ```
   bash
   ./build.sh
   ```
3. **Find your application**:
   - Location: `dist/HumanBlurTool.app`

Method 2: Using PyInstaller with Spec File (Manual)

1. **Navigate to the project directory**:
   ```
   bash
   cd /path/to/human-blur-tool
   ```
2. **Build using the spec file**:
   ```
   bash
   pyinstaller HumanBlurTool.spec --clean
   ```
3. **Find your application**:
   - Location: `dist/HumanBlurTool.app`

Testing the macOS Application

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

Removing Quarantine Attribute

If macOS shows a security warning:

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

Code Signing (for Distribution)

If you plan to distribute the macOS app, you'll need to code sign it:

```
# Sign the app
codesign --deep --force --verify --verbose --sign "Developer ID Application: Your
Name" dist/HumanBlurTool.app

# Verify signature
codesign --verify --deep --strict --verbose=2 dist/HumanBlurTool.app
```

Optional: Create DMG Installer

```
# Install create-dmg
brew install create-dmg

# Create DMG
create-dmg \
  --volname "Human Blur Tool" \
  --window-pos 200 120 \
  --window-size 600 400 \
  --icon-size 100 \
  --app-drop-link 425 120 \
  "HumanBlurTool.dmg" \
  "dist/HumanBlurTool.app"
```

Building on Linux

Method 1: Using the Build Script (Easiest)

1. **Open Terminal** and navigate to the project directory:
   `bash`
   `cd /path/to/human-blur-tool`
2. **Run the build script**:
   `bash`
   `./build.sh`
3. **Find your executable**:
   - Location: `dist/HumanBlurTool`

Method 2: Using PyInstaller with Spec File (Manual)

1. **Navigate to the project directory**:
   `bash`
   `cd /path/to/human-blur-tool`
2. **Build using the spec file**:
   `bash`
   `pyinstaller HumanBlurTool.spec --clean`
3. **Find your executable**:
   - Location: `dist/HumanBlurTool`
4. **Make it executable** (if not already):
   `bash`
   `chmod +x dist/HumanBlurTool`

Testing the Linux Executable

```
./dist/HumanBlurTool
```

What's Fixed

This build configuration fixes the following issues that occurred in previous PyInstaller compilations:

1. ✅ Logo Not Appearing

Problem: The GEN logo (logo.png) didn't show up in the compiled application window.

Solution:
- Implemented `get_resource_path()` function that correctly handles resource paths for both frozen (compiled) and unfrozen (development) modes
- Logo is properly bundled using `--add-data` in the spec file
- The GUI code now uses `sys._MEIPASS` when running as a frozen executable

2. ✅ Processing Hangs/Doesn't Work

Problem: When running the compiled version, processing would say "processing" but never actually process anything - it would just hang.

Solutions Implemented:
- Added `multiprocessing.freeze_support()` in the main entry point
- Created a runtime hook (`runtime_hook.py`) to properly initialize multiprocessing in frozen mode
- Added all necessary hidden imports for ultralytics, PyTorch, OpenCV, and other dependencies
- Set proper multiprocessing start method for frozen executables

3. ✅ Hidden Import Issues

Problem: Some modules weren't being detected by PyInstaller automatically.

Solution: Added comprehensive hidden imports in the spec file:
- PIL and tkinter modules
- All ultralytics and YOLO dependencies
- PyTorch and torchvision components
- OpenCV and numpy components
- HEIC support libraries

4. ✅ Optimized Build Configuration

Benefits:
- Single-file executable for easy distribution
- Windowed mode (no console) for clean GUI experience
- UPX compression for smaller file size
- Proper macOS app bundle with metadata
- Cross-platform build scripts for automation

Technical Details

Resource Path Handling

The code now uses a helper function to correctly locate resources:

```python
def get_resource_path(relative_path):
    """Get absolute path to resource, works for dev and PyInstaller."""
    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
```

Multiprocessing Fix

To ensure PyTorch/ultralytics work in frozen mode:

```python
def main():
    import multiprocessing
    multiprocessing.freeze_support()
    # ... rest of the code
```

Build Configuration Files

- **HumanBlurTool.spec**: Complete PyInstaller configuration with all dependencies
- **runtime_hook.py**: Fixes multiprocessing initialization
- **build.sh**: Automated build script for macOS/Linux
- **build.bat**: Automated build script for Windows

Method 2: AppImage (Cross-Distribution)

AppImage creates a portable executable that works across Linux distributions:

1. **Install python-appimage**:
   `bash`
   `pip install python-appimage`
2. **Create AppImage**:
   `bash`
   `python-appimage build app -l manylinux2014_x86_64 gui.py`

Method 3: Debian Package (.deb)

For Debian/Ubuntu users:

1. **Install fpm** (Effing Package Management):
   `bash`
   `gem install fpm`
2. **Create package structure**:
   ```bash
   mkdir -p package/usr/local/bin
   mkdir -p package/usr/share/applications
   mkdir -p package/usr/share/icons
```

cp dist/HumanBlurTool package/usr/local/bin/
cp logo.png package/usr/share/icons/humanblur.png
```

1. **Create desktop entry** ( `package/usr/share/applications/humanblur.desktop` ):
   `ini`
   `[Desktop Entry]`
   `Name=Human Blur Tool`
   `Comment=AI-Powered Privacy Protection`
   `Exec=/usr/local/bin/HumanBlurTool`
   `Icon=/usr/share/icons/humanblur.png`
   `Terminal=false`
   `Type=Application`
   `Categories=Graphics;Photography;`

2. **Build .deb package**:
   `bash`
   `fpm -s dir -t deb -n humanblur-tool -v 3.0 -C package`

### Testing the Linux Executable

```
./dist/HumanBlurTool
```

Advanced PyInstaller Configuration

For more control, create a `.spec` file:

```
pyinstaller --name "HumanBlurTool" gui.py --windowed --onefile
```

This creates `HumanBlurTool.spec`. Edit it for advanced configuration:
```

```
# -*- mode: python ; coding: utf-8 -*-

block_cipher = None

a = Analysis(
    ['gui.py'],
    pathex=[],
    binaries=[],
    datas=[('logo.png', '.')],
    hiddenimports=[
        'PIL._tkinter_finder',
        'cv2',
        'numpy',
        'ultralytics',
        'torch',
    ],
    hookspath=[],
    hooksconfig={},
    runtime_hooks=[],
    excludes=[],
    win_no_prefer_redirects=False,
    win_private_assemblies=False,
    cipher=block_cipher,
    noarchive=False,
)

pyz = PYZ(a.pure, a.zipped_data, cipher=block_cipher)

exe = EXE(
    pyz,
    a.scripts,
    a.binaries,
    a.zipfiles,
    a.datas,
    [],
    name='HumanBlurTool',
    debug=False,
    bootloader_ignore_signals=False,
    strip=False,
    upx=True,
    upx_exclude=[],
    runtime_tmpdir=None,
    console=False,
    disable_windowed_traceback=False,
    argv_emulation=False,
    target_arch=None,
    codesign_identity=None,
    entitlements_file=None,
    icon='logo.png'
)

# For macOS .app bundle
app = BUNDLE(
    exe,
    name='HumanBlurTool.app',
    icon='logo.png',
    bundle_identifier='org.globalemancipation.humanblur',
    info_plist={
        'NSHighResolutionCapable': 'True',
        'LSMinimumSystemVersion': '10.13.0',
    },
)
```

Build using the spec file:

```
pyinstaller HumanBlurTool.spec
```

Reducing Executable Size

Large executables are common with PyTorch and OpenCV. Here are optimization tips:

1. Exclude Unused Modules

```
pyinstaller --onefile --windowed \
    --exclude-module matplotlib \
    --exclude-module scipy \
    --exclude-module pandas \
    gui.py
```

2. Use UPX Compression

```
# Install UPX
# Windows: Download from https://upx.github.io/
# macOS: brew install upx
# Linux: sudo apt-get install upx

pyinstaller --onefile --windowed --upx-dir=/path/to/upx gui.py
```

3. Strip Debug Symbols (Linux/macOS)

```
pyinstaller --onefile --windowed --strip gui.py
```

Troubleshooting

Issue: Build fails with “ModuleNotFoundError”

Cause: A required module is not included in the hidden imports.

Solution:
1. Check the error message for the missing module name
2. Add it to the `hiddenimports` list in `HumanBlurTool.spec`
3. Rebuild using the spec file

Example:

```
hiddenimports=[
    # ... existing imports ...
    'your_missing_module_here',
]
```

Issue: Logo doesn’t appear in compiled app

Status: ✅ This issue is now fixed in the current configuration.

If you still experience this:
1. Verify `logo.png` exists in the project root
2. Check that the spec file includes: `datas=[('logo.png', '.')]`
3. Ensure `gui_enhanced.py` uses `get_resource_path("logo.png")`

Issue: Processing hangs or doesn’t work

Status: ✅ This issue is now fixed in the current configuration.

If you still experience this:
1. Verify `multiprocessing.freeze_support()` is called in `main()`
2. Check that `runtime_hook.py` is included in the spec file
3. Enable console mode temporarily to see error messages:
- In `HumanBlurTool.spec`, change `console=False` to `console=True`
- Rebuild and check the console output

Issue: YOLO models not downloading

Solution: Models are downloaded automatically by ultralytics on first use. The app needs internet access the first time it runs to download the selected model (yolov8n-seg.pt by default, ~6MB).

For offline distribution: Pre-download and bundle models:

1. Download models:
```
python
   from ultralytics import YOLO
   YOLO('yolov8n-seg.pt')
   YOLO('yolov8s-seg.pt')
   YOLO('yolov8m-seg.pt')
```
2. Find model location (usually `~/.cache/ultralytics/`)
3. Copy models to project and update spec file:
```
python
   datas=[
       ('logo.png', '.'),
       ('models/*.pt', 'models'),
   ]
```
4. Update code to load from bundled location

Issue: Executable is too large (>500MB)

Cause: PyTorch and dependencies are large.

Solutions:

1. **Use CPU-only PyTorch** (recommended if no GPU needed):
```
bash
   pip uninstall torch torchvision
   pip install torch torchvision --index-url https://download.pytorch.org/whl/cpu
```
This can save 100-200 MB.

2. **Create directory-based build instead of single file**:

In `HumanBlurTool.spec`, change the EXE section:

```python
exe = EXE(
pyz,
a.scripts,
# Comment out or remove these lines:
# a.binaries,
# a.zipfiles,
# a.datas,
# Keep only:
[],
exclude_binaries=True, # Add this line
name='HumanBlurTool',
# ... rest of the config
)

# Add this after EXE:
coll = COLLECT(
exe,
a.binaries,
a.zipfiles,
a.datas,
strip=False,
upx=True,
upx_exclude=[],
name='HumanBlurTool'
)
```

This creates a folder with the executable and dependencies (faster startup, slightly larger total).

1. **Exclude unnecessary modules**: Already done in the provided spec file.

Issue: macOS "App is damaged and can't be opened"

Cause: Gatekeeper security on macOS blocks unsigned apps.

Solution: Remove quarantine attribute:

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

For distribution: Code sign the app (requires Apple Developer account):

```
codesign --deep --force --verify --verbose --sign "Developer ID Application: Your Name" dist/HumanBlurTool.app
```

Issue: Windows SmartScreen warning

Cause: Unsigned executable from unknown publisher.

Solution for users: Click "More info" → "Run anyway"

For distribution: Code sign the executable (requires code signing certificate)

Issue: Slow startup time

Cause: Large dependencies (PyTorch, OpenCV) being loaded.

Solutions:
1. Use directory-based build instead of `--onefile` (faster extraction)
2. This is expected behavior - first launch extracts files
3. Subsequent launches should be faster

Issue: Build script fails on macOS/Linux

Error: `Permission denied` or `command not found`

Solution:

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

Issue: “ImportError: DLL load failed” on Windows

Cause: Missing Visual C++ redistributables.

Solution: Install Microsoft Visual C++ Redistributables:
- Download from: https://aka.ms/vs/17/release/vc_redist.x64.exe
- Or include them with your installer

Distribution Checklist

Before distributing your executable:

- [ ] Test on a clean machine without Python installed
- [ ] Test all features (file selection, folder processing, help dialog)
- [ ] Verify logo displays correctly
- [ ] Check error messages are user-friendly
- [ ] Include README with system requirements
- [ ] Test on the target OS version
- [ ] For macOS: Code sign if possible
- [ ] For Windows: Consider creating an installer
- [ ] Include LICENSE file
- [ ] Document any third-party dependencies

File Size Expectations

Expected file sizes (approximate):

- **Windows**: 400-800 MB (with PyTorch)
- **macOS**: 450-900 MB (with PyTorch)

- **Linux**: 400-800 MB (with PyTorch)

CPU-only PyTorch builds are typically 50-100 MB smaller.

Additional Resources

- PyInstaller Documentation (https://pyinstaller.org/en/stable/)
- Ultralytics YOLOv8 Documentation (https://docs.ultralytics.com/)
- Python Packaging Guide (https://packaging.python.org/)
- Inno Setup (Windows Installer) (https://jrsoftware.org/isinfo.php)
- create-dmg (macOS DMG Creator) (https://github.com/create-dmg/create-dmg)

Support

For build issues or questions about distribution, please contact:
apps@globalemancipation.ngo