torch-check

Inspect your Linux/Python/NVIDIA environment and recommend a compatible, officially available PyTorch wheel.

Published: Aug 08, 2026

`torch-check` inspects a Linux/Python/NVIDIA environment, enumerates wheels that actually exist on the official PyTorch indexes, and explains which PyTorch configuration can be installed and run.

It deliberately does **not** treat the `CUDA Version` field printed by `nvidia-smi` as a strict upper bound. Compatibility is based on the installed driver, NVIDIA's CUDA major-family minor-version compatibility rules, the exact wheel tags, glibc, and conservatively maintained GPU-architecture evidence.

Supported production target

- Linux x86_64 with glibc
- CPython
- NVIDIA GPUs, plus an explicit CPU-wheel fallback
- Official PyTorch pip indexes
- `pip`, `uv pip`, and `uv add` command generation

Windows, macOS, Arm64, musl, ROCm, XPU, Conda, source builds, third-party CUDA extensions, and container/host split-driver analysis are detected without a panic but are not currently resolved as supported targets.

On musl hosts, resolution fails closed with `unsupported_libc`; the static musl release binary still inspects a glibc host at runtime, but it never recommends a manylinux wheel for a musl host.

Install

The POSIX shell release installer supports Linux, macOS, and Windows under Git Bash/MSYS/Cygwin. It selects the archive for the current operating system and architecture, verifies it against the release `SHA256SUMS`, and atomically installs the binary to the user-local `$HOME/.local/bin` directory without invoking `sudo` or modifying shell startup files. Native PowerShell and Command Prompt users should install the Windows archive from GitHub Releases directly.

```
curl -fsSL https://raw.githubusercontent.com/rice8y/torch-check/main/scripts/install.sh | sh
```

Pass options after `sh -s --`, for example `curl -fsSL https://raw.githubusercontent.com/rice8y/torch-check/main/scripts/install.sh | sh -s -- --version 0.1.1` or `curl -fsSL https://raw.githubusercontent.com/rice8y/torch-check/main/scripts/install.sh | sh -s -- --install-dir "$HOME/bin"`; the equivalent `TORCH_CHECK_VERSION` and `TORCH_CHECK_INSTALL_DIR` environment variables are also supported. Run a downloaded copy with `--help` for the complete interface. Check the final warning and add the selected directory to `PATH` when necessary.

From crates.io:

```
cargo install torch-check --locked
```

From a source checkout:

```
cargo install --path . --locked
```

Prebuilt archives are published through GitHub Releases:

Platform	Rust target	Validation and support
Ubuntu 22.04.x x86_64, including 22.04.2	`x86_64-unknown-linux-musl`	Native Ubuntu 22.04 smoke test; static musl binary; full production recommendation support

Ubuntu 22.04 ARM64	`aarch64-unknown-linux-musl`	Native smoke test; inspection only
macOS Intel	`x86_64-apple-darwin`	Native test on macOS 15; inspection only; deployment target 10.15
macOS Apple Silicon	`aarch64-apple-darwin`	Native test on macOS 15; inspection only; deployment target 11.0
Windows x64	`x86_64-pc-windows-msvc`	Native test on Windows Server 2022; inspection only; static CRT

Every archive contains the binary, shell completions, a man page, project licenses, the security policy, and target-specific third-party license notices. `SHA256SUMS` covers every archive. The non-Linux-x86_64 binaries can start and inspect their host, but PyTorch wheel recommendation remains intentionally limited to the production target described above.

Usage

```
# Alias for `recommend`
torch-check
torch-check --installer uv
torch-check --torch-version '>=2.10,<3'
torch-check --with torchvision,torchaudio

torch-check inspect
torch-check recommend
torch-check recommend --installer uv
torch-check recommend --installer uv-add
torch-check recommend --torch-version '>=2.10,<3'
torch-check recommend --with torchvision,torchaudio
torch-check candidates
torch-check candidates --unverified
torch-check candidates --all
torch-check explain torch==2.12.1 --cuda cu130
torch-check verify

torch-check recommend --format json
torch-check --offline candidates
torch-check --refresh recommend
torch-check --python /opt/venv/bin/python recommend
torch-check --gpu 0,2 recommend
```

Generated installation commands are bound to the interpreter whose tags were evaluated. `pip` and `uv add` commands pin that interpreter explicitly; `uv pip` omits `--python` only when `torch-check` has identified the exact existing environment that uv will select—an active PEP 405 environment, an active Conda environment, or the nearest `.venv`—and no uv Python override is present. When `--python` is supplied, the same interpreter is always used for detection, verification, and command generation. Commands are represented internally as an executable and argument vector; the displayed shell command is never executed by `torch-check`.

`--gpu` accepts physical indices from `nvidia-smi`, not CUDA-visible ordinals. During verification, `torch-check` selects devices by UUID and reports the physical-to-logical mapping used by the isolated Python process. A numeric `CUDA_VISIBLE_DEVICES` subset is rejected when that mapping cannot be established safely.

Human output adapts to the terminal width, groups identical selected GPUs, and uses restrained status colors only when stdout is a terminal. Set `NO_COLOR` or `CLICOLOR=0` to disable ANSI styling; redirected output and JSON never contain color escapes. The default recommendation is intentionally summarized. `torch-check candidates` lists only `verified`, `direct-compatible`, and `minor-compatible` results. Add `--unverified` to include candidates that still need release-configuration review, static evidence, or runtime verification, or use `--all` for every compatibility status and exclusion reason. Explicit `--torch-version` constraints remain hard filters in every mode.

Pinned command forms are:

```
/path/to/detected/python -m pip install --isolated torch==VERSION \
  --index-url https://download.pytorch.org/whl/VARIANT

uv pip install --python /path/to/detected/python torch==VERSION \
  --default-index https://download.pytorch.org/whl/VARIANT

uv add --python /path/to/detected/python torch==VERSION \
  --index pytorch=https://download.pytorch.org/whl/VARIANT
```

When an existing `uv pip` target is established without ambiguity, the shorter environment-relative form is emitted:

```
uv pip install torch==VERSION \
  --default-index https://download.pytorch.org/whl/VARIANT
```

What the states mean

The JSON report retains independent results for wheel existence, Python/ABI, platform/glibc, GPU architecture, NVIDIA driver, and runtime verification. The human-facing aggregate is derived from those checks:

- `verified`: `torch-check verify` executed the installed build successfully on every selected logical GPU in this invocation; when NVIDIA GPUs are detected and `--gpu` is omitted, every detected GPU is selected by UUID, so a CPU-only build does not pass as GPU-verified.
- `direct-compatible`: all static checks pass and the driver meets the normal minimum for the wheel's exact CUDA release.
- `minor-compatible`: all static checks pass, but the result relies on NVIDIA's same-major CUDA minor-version compatibility.
- `unverified`: no known condition rules the wheel out, but trustworthy evidence is missing (most often static GPU architecture coverage).
- `incompatible`: a known platform, GPU, or driver condition fails.
- `unavailable`: the required official wheel/ABI does not exist or is yanked.

The default recommendation is intentionally stricter than candidate discovery. A CUDA wheel is recommendation-eligible only when it belongs to the reviewed PyTorch release configuration and its static result is `direct-compatible`, `minor-compatible`, or `verified`. Index wheels with incomplete architecture evidence, and index-only variants absent from the reviewed release matrix, are available through `torch-check candidates --unverified` but never receive a default install command. On a detected NVIDIA system, `torch-check` searches every matching release series and prefers the newest recommendation-eligible CUDA wheel even when a newer CPU wheel exists. Only when no reviewed CUDA candidate passes every static check does it fall back to the newest reviewed direct-compatible CPU wheel and explain why the CUDA candidates need verification or a driver upgrade.

Minor-version compatibility has documented limitations. PTX JIT and operations that depend on features added by a newer driver may still require a driver upgrade. `torch-check` reports that limitation instead of silently treating a minor-compatible result as guaranteed.

The local CUDA Toolkit is displayed because it matters for source builds and extensions. Its absence does not exclude an official PyTorch CUDA wheel, which ships its runtime dependencies separately.

Metadata and cache

The wheel list is discovered from `https://download.pytorch.org/whl/` and the simple indexes below it. Links are accepted only over HTTPS from the exact PyTorch download host allowlist. A recommendation is produced only from a complete metadata snapshot; a partially fetched set is rejected.

Snapshots are cached for 24 hours in the platform cache directory (normally `$XDG_CACHE_HOME/torch-check/` or `~/.cache/torch-check/` on Linux). Cache files are keyed by the requested package set, so a torch-only refresh cannot destroy an offline-capable torchvision/torchaudio snapshot. Writes are locked, atomic, and monotonic: a slower concurrent refresh cannot replace an equally recent or newer snapshot.

- `--refresh` bypasses a fresh cache.
- `--offline` performs no network request and requires an existing complete cache.

- A stale cache may be used after a network failure, but the report identifies it as `stale_if_error` and warns about its age.

CUDA driver rules and official release preferences are reviewed data files in `data/`. Their source URLs and review date are stored with the data. The scheduled bounded observer records PyTorch releases/architectures and NVIDIA driver tables in a content-addressed pull request. It never changes compatibility rules automatically; CI requires a human-reviewed rule update before a changed observation can be merged.

Static GPU architecture evidence is currently maintained for the PyTorch 2.6–2.13 releases listed in the reviewed data. PyTorch 2.6–2.11 coverage is reviewed against tag-pinned, content-hashed Linux wheel build scripts, including each exact stable patch tag recorded in the reviewed data. PyTorch 2.12–2.13 coverage is reviewed against the current release matrix. Rules name exact public versions, so a new patch release never inherits older architecture evidence implicitly. Wheels from PyTorch 2.5 and earlier remain `unverified`: their tagged workflows select mutable external builder branches, so the exact builder checkout used for an artifact cannot be established from tag source alone.

Primary sources:

- NVIDIA CUDA minor-version compatibility
- NVIDIA CUDA release driver table
- PyTorch official wheel index
- PyTorch release matrix
- Previous PyTorch versions
- PyTorch 2.6.0 Linux wheel build configuration
- PyTorch 2.7.0 Linux wheel build configuration
- PyTorch 2.7.1 Linux wheel build configuration
- PyTorch 2.8.0 Linux wheel build configuration
- PyTorch 2.9.0 Linux wheel build configuration
- PyTorch 2.9.1 Linux wheel build configuration
- PyTorch 2.10.0 Linux wheel build configuration
- PyTorch 2.11.0 Linux wheel build configuration

JSON contract

Every JSON response contains `"schema_version": 1`. Reason and warning fields use stable codes and parameters rather than terminal prose. Successful command schemas and the common error envelope are in `data/schemas/`. Complete JSON reports, including structured diagnostics, are written to stdout. Human-mode errors and failures to serialize a JSON document are written to stderr; ordinary JSON-mode failures still produce the error envelope on stdout.

Environment output can contain device names, GPU UUIDs, and interpreter paths. Use `--redact` before sharing a report.

Verification and trust boundary

Environment inspection and recommendation start the selected interpreter with `-I -S` and execute bounded system probes such as `uname`, `ldd`, `nvidia-smi`, and `nvcc` from `PATH` or the configured `CUDA_HOME`. `torch-check verify` additionally imports the installed `torch` package. Use all commands only with a trusted interpreter, filesystem, `PATH`, and `CUDA_HOME`; importing an installed Python package executes its code. Verification does not install, remove, or update anything. For a CUDA build it exercises each selected logical device with allocation, elementwise arithmetic, matrix multiplication, and synchronization. cuDNN availability is reported but is not by itself a failure.

Static recommendations never receive the `verified` state.

Exit codes

Code	Meaning
0	Successful result without warnings

1	Valid result with warnings (including minor-compatible/unverified or a CPU fallback on a detected NVIDIA system)
2	Incompatible result, failed verification, or CLI usage error
3	Required network/index/cache metadata unavailable
4	Unexpected internal failure

Privacy and security

`torch-check` has no telemetry and does not upload detected environment data. Network access is limited to downloading public package metadata. It never runs an installation command. External probes use argument arrays rather than a shell, with time and output limits. See SECURITY.md for reporting issues.

Development

```
cargo fmt --all -- --check
cargo clippy --workspace --all-targets --all-features --locked -- -D warnings
cargo test --workspace --all-features --locked
shellcheck scripts/install.sh scripts/tests/test_install.sh
scripts/tests/test_install.sh
python3 -m unittest discover -s scripts/tests -v
python3 scripts/check_reviewed_metadata.py
cargo package --locked
```

Normal tests are fixture/mock based and do not require a GPU or network. The scheduled live-index check is separate so upstream drift cannot make pull-request tests nondeterministic.

License