WheelNext Open-Source Initiative

Barry Warsaw
Vyas Ramasubramani
Jonathan Dekhtiar

~

April 2025

01 | PyCon Plan

WheelNext presence at PyCon 2025

- Language Summit (Wed, May 14)
  - PEP 772 - the packaging governance process
  - What about lightning talks?
- Packaging Summit (Sat, May 17)
  - PEP 772 proposed - Barry
  - Wheel Variants - JD
  - Request to attend closes today (April 30th)
- Conference talks
  - Reinventing the Wheel (Sun, May 18, 1-1:30 ET); JD & Barry
  - Introducing WheelNext to a wider audience, PEP review, Call to Action
- Sprints (Mon May 19 - Thu, May 22)
  - JD-led Variants sprint
  - Other packaging sprints?

<Slide Title>

Talks

Reinventing the Wheel: A Community-Driven Roadmap for Python Packaging

Sunday, May 18th, 2025 1 p.m.–1:30 p.m. in Ballroom BC

Presented by Jonathan Dekhtiar, Barry Warsaw

Experience Level: Advance experience

02 Native Lib Loader

Using Shared Libraries Across Wheel Boundaries

Problem: How can we safely reuse shared libraries from other wheels?

- Manylinux-style bundling or static linking balloon package sizes
  - One of the core goals of wheel variants is to help shrink packages
- Related ideas:
  - PEP 725: How to specify external dependencies
  - Wheel Variants: How to ship different builds for a package
- Open question: Since pip does not install binaries to system prefix directories, how can we make libraries available to other packages?

Using Shared Libraries Across Wheel Boundaries

Prior Art

- Existing packages use RPATHs, LD_PRELOAD, add_dll_directory
  - Not portable across platforms, not robust to layout changes (e.g. editable installs), requires consumers to know provider file layouts
- Pynativelib proposal (https://github.com/njsmith/wheel-builders/blob/pynativelib-proposal/)
  - API is attractive, but implementation (environment variables) does not work
- https://github.com/pypackaging-native/dynamic-library
  - Developed concurrently with my work, very similar idea

Using Shared Libraries Across Wheel Boundaries

Goals: https://github.com/wheelnext/native_lib_loader/

- Safe, portable, and robust way to load libraries from other wheels
- Should support
  - Packages declaring exported libraries
  - Packages using libraries from wheels that do not yet advertise their libraries
- Should support runtime checks for whether the library was loaded from the host
- Possible convenience facilities:
  - Use entry points to make available in a global registry
  - Use pth files to automatically preload libraries, maximizing the chances that they are not clobbered by preexisting host libraries
  - Preload locations

03 | Wheel Variant

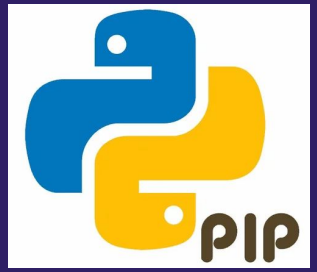
Key Contributors

VariantLib

Michał Górny - Quansight

Jonathan Dekhtiar - NVIDIA

Wheel Variants: Now & Today ! We are live



** Functional in WheelNext Forks - Not official implementations/products **

Very soon... Variant support is expanding!

PyTorch

NumPy

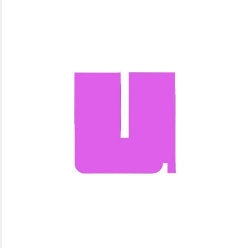
DMLC XGBoost

MESON

pip

NVIDIA CUDA

(Hopefully) Very Soon - Work in Progress



Poetry

Reference Implementation: VariantLib

https://github.com/wheelnext/variantlib/

- A library supporting the (future) Wheel Variant PEP.
- Not part of the standard - **reference implementation**.
  However: we really don't recommend re-implementing VL.
- Almost entirely a "low level" library.
  => End-users: should virtually never interact with `variantlib`
- **Provides:** data model, data validation, resolver, CLI tooling, plugin interfaces, etc.
- **Leveraged by:** package installers, build backend/frontend, package indexes, workflow managers, etc.

A common vocabulary

https://github.com/wheelnext/variantlib/

- **Variant Property:** the exact variant descriptor.
  Example: `x86_64 :: level :: v4`

- **Variant Feature:** a “family” of mutually exclusive variant properties
  Example: `x86_64 :: level`

- **Variant Namespace:** Examples: `x86_64, nvidia, rocm, etc`

- **Variant Description:** Complete and exhaustive description of one single Wheel Variant containing 1 or many variant properties.
  Example:
  - *`x86_64 :: level :: v4`*
  - *`nvidia :: driver :: 12.8`*

Wheel Variant Provider: Plugin Interface


```python
from variantlib.models.provider import VariantFeatureConfig
from variantlib.protocols import PluginType
from variantlib.protocols import VariantFeatureConfigType
from variantlib.protocols import VariantPropertyType

class MyCustomPlugin(PluginType):
    namespace = "my_custom_namespace"

    def get_all_configs(self) -> list[VariantFeatureConfigType]:
        return [
            VariantFeatureConfig("feature1", ["value1", "value2", "value3"]),
            VariantFeatureConfig("feature2", ["valueA", "valueB"]),
        ]

    def get_supported_configs(self) -> list[VariantFeatureConfigType]:
        # Return a subset of configurations that are supported on this system
        return [
            VariantFeatureConfig("feature1", ["value1", "value2"]),
            VariantFeatureConfig("feature2", ["valueA"]),
        ]

    def get_build_setup(self, properties: list[VariantPropertyType]) -> dict[str, list[str]]:
        # Provide compiler flags to build-backends (if necessary)
        env_vars = {}
        for prop in properties:
            assert prop.namespace == self.namespace
            if prop.feature == "feature1":
                env_vars["cflags"] = f"-march=Custom{prop.value}"
        return env_vars
```

Wheel Variant Provider: Plugin Interface


```python
from variantlib.models.provider import VariantFeatureConfig
from variantlib.protocols import PluginType
from variantlib.protocols import VariantFeatureConfigType
from variantlib.protocols import VariantPropertyType

class MyCustomPlugin(PluginType):
    namespace = "my_custom_namespace"

    def get_all_configs(self) -> list[VariantFeatureConfigType]:
        return [
            VariantFeatureConfig("feature1", ["value1", "value2", "value3"]),
            VariantFeatureConfig("feature2", ["valueA", "valueB"]),
        ]

    def get_supported_configs(self) -> list[VariantFeatureConfigType]:
        # Return a subset of configurations that are supported on this system
        return [
            VariantFeatureConfig("feature1", ["value1", "value2"]),
            VariantFeatureConfig("feature2", ["valueA"]),
        ]

    def get_build_setup(self, properties: list[VariantPropertyType]) -> dict[str, list[str]]:
        # Provide compiler flags to build-backends (if necessary)
        env_vars = {}
        for prop in properties:
            assert prop.namespace == self.namespace
            if prop.feature == "feature1":
                env_vars["cflags"] = f"-march=Custom{prop.value}"
        return env_vars
```

```python
from typing import Protocol
```

We do not enforce any dependency/inheritance

Variant User Preferences/Configuration


```
$ variantlib config setup

-------------------------------------------------------------------------------#
#                                 INSTRUCTIONS                                  #
#                                                                               #
# This command will set the variant configuration file for the requested       #
# environment. The configuration file is used to adjust the priority order     #
# of variants among the compatible variants on the system.                      #
#                                                                               #
-------------------------------------------------------------------------------

- Namespace Priorities [REQUIRED]:
    If more than one plugin is installed, this setting is mandatory to
    understand which variant namespace goes first.
    Example: `MPI` > `x86_64`
    ~ Meaning that `variantlib` will prioritize MPI support over special
      x86_64 optimizations. ~

-------------------------------------------------------------------------------

...
```

Variant User Preferences/Configuration


```
──────────────────────── namespace_priorities ────────────────────────
Please order the namespaces according to their priority, most preferred namespace
first. Press [Page Up or Tab] and Page Down to reorder the focused namespace, and
enter to toggle it. Only enabled namespaces will be included in the configuration.
The namespacs in bold are required and cannot be disabled.
[X] nvidia
[X] x86_64
< Save                                                                        >
< Abort                                                                       >
```

** We expect tools / installers / workflow managers to create their own interface in the future **

For now, please enjoy that state-of-the-art 1990s design.

Variant User Preferences/Configuration: `variants.toml`


```toml
# ============= Top-Most Priority ============= #

# 1. Define the priority of variant properties - 1st order priority.
#    => Expected format: "namespace::feature::value"

property_priorities = [
    "fictional_hw::architecture::mother",
    "fictional_tech::risk_exposure::25",
]

# ============= Second-Most Priority ============= #

# 2. Define the priority of variant features - 2nd order priority under the variant properties.
#    => Expected format: "namespace::feature"

feature_priorities = [
    "fictional_hw::architecture",
    "fictional_tech::risk_exposure",
]

# ============= Default Priority Ordering ============= #

# 3. Define the priority of variant namespaces
#    => Expected format: "namespace"

namespace_priorities = [
    "fictional_hw",
    "fictional_tech",
]
```

Variant Resolver: “Finding the best Variant”


```
# ============= Top-Most Priority ============= #

# 1. Define the priority of variant properties - 1st order priority.
#    => Expected format: "namespace::feature::value"

property_priorities = [
    "fictional_hw::architecture::mother",
    "fictional_tech::risk_exposure::25",
]
```

Dimension 1

```
# ============= Second-Most Priority ============= #

# 2. Define the priority of variant features - 2nd order priority under the variant properties.
#    => Expected format: "namespace::feature"

feature_priorities = [
    "fictional_hw::architecture",
    "fictional_tech::risk_exposure",
]
```

Dimension 2

```
# ============= Default Priority Ordering ============= #

# 3. Define the priority of variant namespaces
#    => Expected format: "namespace"

namespace_priorities = [
    "fictional_hw",
    "fictional_tech",
]
```

Dimension 3

Variant Resolver


```python
def sort_and_filter_supported_variants(
    vdescs: list[VariantDescription],
    supported_vprops: list[VariantProperty],
    namespace_priorities: list[str] | None = None,
    feature_priorities: list[VariantFeature] | None = None,
    property_priorities: list[VariantProperty] | None = None
) -> list[VariantDescription]:

    # Step 1: we remove any duplicate, or unsupported `VariantDescription` on
    #         this platform.
    filtered_vdescs = filter_variants(
        vdescs=vdescs,
        allowed_properties=supported_vprops,
        forbidden_namespaces=forbidden_namespaces,
        forbidden_features=forbidden_features,
        forbidden_properties=forbidden_properties,
    )

    ...


    # Step 2: we sort the `VariantDescription` based on the sorted supported properties
    #         and their respective priority.
    return sort_variants_descriptions(
        filtered_vdescs,
        property_priorities=sorted_supported_vprops,
    )
```

Variant Resolver


```python
def sort_and_filter_supported_variants(
    vdescs: list[VariantDescription],
    supported_vprops: list[VariantProperty],
    namespace_priorities: list[str] | None = None,
    feature_priorities: list[VariantFeature] | None = None,
    property_priorities: list[VariantProperty] | None = None
) -> list[VariantDescription]:

    # Step 1: we remove any duplicate, or unsupported `VariantDescription` on
    #         this platform.
    filtered_vdescs = filter_variants(
        vdescs=vdescs,
        allowed_properties=supported_vprops,
        forbidden_namespaces=forbidden_namespaces,
        forbidden_features=forbidden_features,
        forbidden_properties=forbidden_properties,
    )

    ...


    # Step 2: we sort the `VariantDescription` based on the sorted supported properties
    #         and their respective priority.
    return sort_variants_descriptions(
        filtered_vdescs,
        property_priorities=sorted_supported_vprops,
    )
```

Variant Resolver


```python
def sort_and_filter_supported_variants(
    vdescs: list[VariantDescription],
    supported_vprops: list[VariantProperty],
    namespace_priorities: list[str] | None = None,
    feature_priorities: list[VariantFeature] | None = None,
    property_priorities: list[VariantProperty] | None = None
) -> list[VariantDescription]:

    # Step 1: we remove any duplicate, or unsupported `VariantDescription` on
    #         this platform.
    filtered_vdescs = filter_variants(
        vdescs=vdescs,
        allowed_properties=supported_vprops,
        forbidden_namespaces=forbidden_namespaces,
        forbidden_features=forbidden_features,
        forbidden_properties=forbidden_properties,
    )

    ...

    # Step 2: we sort the `VariantDescription` based on the sorted supported properties
    #         and their respective priority.
    return sort_variants_descriptions(
        filtered_vdescs,
        property_priorities=sorted_supported_vprops,
    )
```

Variant Resolver: 3-Dimensional Sorting


```
# ============================== variants.toml ============================== #

# Defines the priority of variant namespaces
#    => Expected format: "namespace"

namespace_priorities = [
    "fictional_hw",
    "fictional_tech",
]
```

- **Namespace A > Namespace B**
  Example: `gpu` > `x86_64`

The user prefers a GPU-accelerated build over a build optimized for their CPU.

Variant Resolver: 3-Dimensional Sorting


```
# ============================== variants.toml ============================== #

# Defines the priority of variant features
#    => Expected format: "namespace::feature"

feature_priorities = [
    "fictional_hw::architecture",
    "fictional_tech::risk_exposure",
]
```

- **Variant Feature A > Variant Feature B**
  Example: `x86_64 :: level` > `x86_64 :: avx512f`

The user prefers a wheel for his CPU x86_64 level

Variant Resolver: 3-Dimensional Sorting


```
# ============================== variants.toml ============================== #

# Defines the priority of variant properties
#    => Expected format: "namespace::feature::value"

property_priorities = [
    "fictional_hw::architecture::mother",
    "fictional_tech::risk_exposure::25",
]
```

- **Variant Property A > Variant Property B**
  Example: `nvidia :: driver :: 12.8` > `nvidia :: driver :: 12.6`

The user prefers a wheel for the latest CUDA driver

Variant Resolver: 3-Dimensional Sorting

https://github.com/wheelnext/variantlib/

- **Total Execution Time: < 50ms with over 1000 variants**
- **Requires a variant-enabled Python Package Index: `variants.json`**

```
{
    "providers": {
        "nvidia": ["nvidia-variant-provider"]
    },
    "variants": {
        "73be618d": {
            "nvidia": {
                "driver": "11"
            }
        },
        "aa00cb06": {
            "nvidia": {
                "driver": "12"
            }
        }
    }
}
```

VariantLib CLI - Tooling & Debugging

How to Build a Wheel Variant ?

```
$ variantlib make-variant \
    --file xgboost-3.1.0-py3-none-win_amd64.whl \
    --property "nvidia :: driver :: 12" \
    --property "nvidia :: arch :: sm90" \
    --property "x86_64 :: level :: v4" \
    --output-directory .

Variant Created: xgboost-3.1.0-py3-none-win_amd64-a0e2749e.whl
```

VariantLib CLI - Tooling & Debugging

How to Build a Wheel Variant ?

```
$ python -m build -w -Cvariant=x86_64::level::v3
* Creating isolated environment: venv+pip...
  - meson-python @ https://github.com/wheelnext/meson-python/archive/wheel-variants.tar.gz
  - variant_x86_64 @ https://github.com/wheelnext/variant_x86_64/archive/main.tar.gz

* Getting build dependencies for wheel...
* Building wheel...

CPU Optimization Options
  baseline:
    Requested : AVX2
    Enabled   : SSE SSE2 SSE3 SSSE3 SSE41 POPCNT SSE42 AVX F16C FMA3 AVX2

Successfully built numpy-2.2.5-cp312-cp312-linux_x86_64-fa7c1393.whl
```

VariantLib CLI - Tooling & Debugging

How to analyze your platform ?


```
$ variantlib analyze-platform

variantlib.loader - INFO - Discovering Wheel Variant plugins...
variantlib.loader - INFO - Loading plugin from entry point: x86_64
variantlib.loader - INFO - Loading plugin from entry point: nvidia_variant_provider

#################### Provider Config: `nvidia` ####################
    - Variant Config [001]: driver :: ['12.8', '12.7', ... '12.0', '12']
###################################################################

#################### Provider Config: `x86_64` ####################
    - Variant Config [001]: level :: ['v4', 'v3', 'v2', 'v1']
###################################################################
```

VariantLib CLI - Tooling & Debugging

How to analyze your platform ?

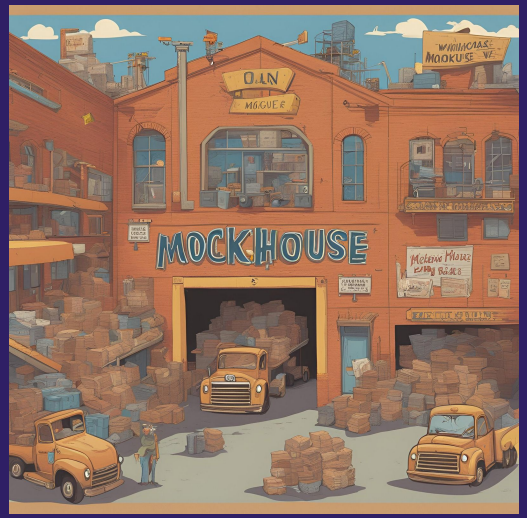

```
$ variantlib get-supported-configs

variantlib.loader - INFO - Discovering Wheel Variant plugins...
variantlib.loader - INFO - Loading plugin from entry point: x86_64
variantlib.loader - INFO - Loading plugin from entry point: nvidia_variant_provider

nvidia :: driver :: 12.8
...
nvidia :: driver :: 12.0
nvidia :: driver :: 12
x86_64 :: level :: v4
x86_64 :: level :: v3
x86_64 :: level :: v2
x86_64 :: level :: v1
```

Variant Package Index: MockHouse

mockhouse = unittest.mock("https://github.com/pypi/warehouse.git")



https://variants-index.wheelnext.dev/

Variant Package Index: MockHouse

https://variants-index.wheelnext.dev/

MockHouse

A fast development platform for the WheelNext Open-Source Initiative.

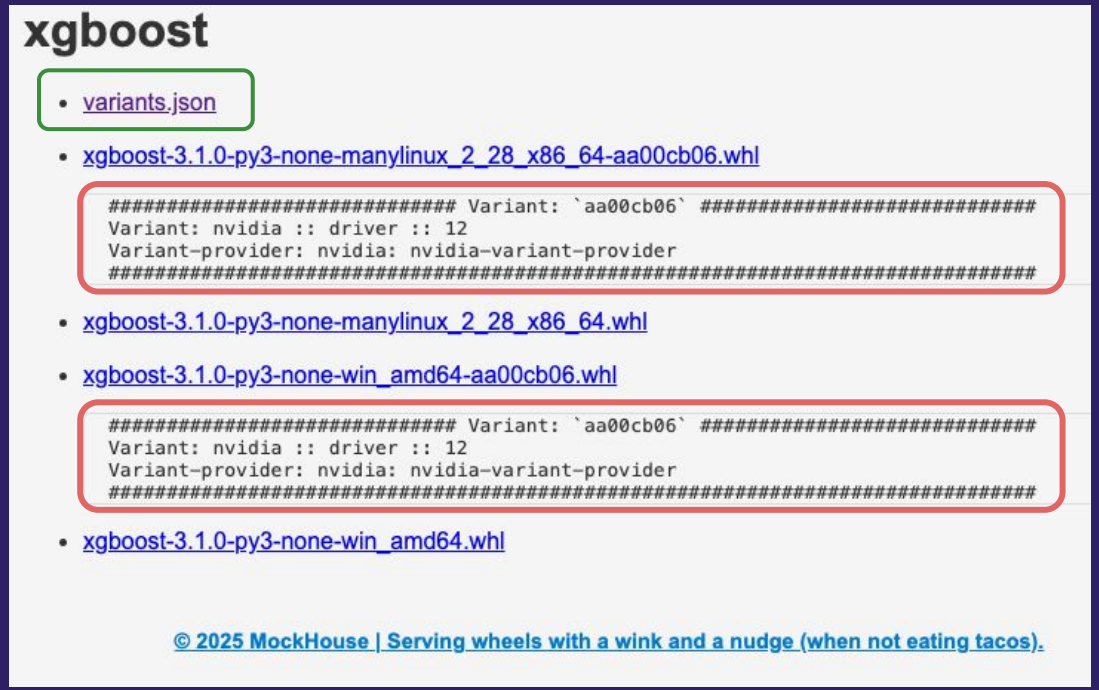
(Almost) Simple Index

This page only highlights the top level packages. Dependencies are accessible - just not listed.

- numpy
- nvidia-cublas
- nvidia-cuda-cupti
- nvidia-cuda-nvrtc
- nvidia-cuda-runtime
- nvidia-cudnn
- nvidia-cufft
- nvidia-cufile
- nvidia-curand
- nvidia-cusolver
- nvidia-cusparse
- nvidia-cusparselt
- nvidia-nccl
- nvidia-nvjitlink
- nvidia-nvtx
- nvidia-variant-provider
- pep-xxx-wheel-variants
- torch
- torchaudio
- torchvision
- variant-x86-64
- xgboost

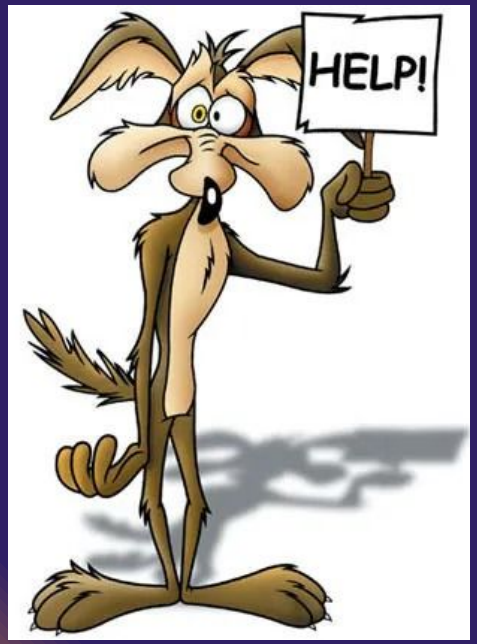
Variant Package Index: MockHouse

https://variants-index.wheelnext.dev/



Community Feedback !

https://github.com/wheelnext/pep_xxx_wheel_variants



- **10/15 min: Please try our tutorial (README.md) and provide feedback: github.com/wheelnext/pep_xxx_wheel_variants**
- **PyCON: Let us know if you will be coming !** Let's chat
- **Help us to:**
  - Find cases where it doesn't work / is insufficient.
  - Integrate Wheel Variants in the different:
    - Installers
    - Build Backend/Frontend
    - Core community packages.

We crucially need feedback to proceed with the PEP !

https://github.com/wheelnext — Contribute

https://wheelnext.dev — Participate

https://discuss.python.org/c/packaging/ — Let's engage

discord.com/channels/803025117553754132/ — Join us on Discord