Vakint

Single-scale vacuum-integral matching and evaluation

---

1 Overview

Vakint matches single-scale vacuum integrals to a library of known topologies, canonicalizes their momentum routing, reduces tensor numerators, and evaluates them analytically or numerically. Expressions are represented with Symbolica and contain a numerator multiplied by a `topo(...)` structure built from propagators.

> **Separate matching from evaluation**
> Canonical topology matching is useful on its own and is much cheaper than a complete evaluation. Tensor reduction and analytic backends invoke FORM; numerical sector decomposition invokes pySecDec and FORM. Choose the narrowest operation and evaluation order needed for the task.

1.1 Choose a task

- To install Symbolica and canonicalize an integral without external tools, follow the Python guide and compare with the generated topology table.
- To compile the same matching-only workflow in Rust, use the Rust guide, then continue with the matching tutorial.
- To tensor-reduce and evaluate with an explicit backend policy, use the evaluation guide with the exact `EvaluationOrder` reference.
- To embed Vakint in Rust or `symbolica.community.vakint`, use the interface guide, native Rustdoc, and generated Python signatures before selecting external tools.

1.2 Workflow

A complete calculation proceeds through explicit stages:

- create one `Vakint` engine and reuse it, because topology setup has a cost;
- parse or construct a Symbolica expression in Vakint's namespace;
- canonicalize the topology and momentum routing;
- tensor-reduce the numerator when required;
- evaluate the integral with the first supported backend in the configured order;
- substitute numerical parameters and external momenta when a numerical result is required.

`VakintExpression` is the structured sum of numerator/topology terms. A recognized topology can be rendered in canonical long or short form. If unknown integrals are allowed, matching may preserve them, but evaluation still needs a backend that supports the resulting topology.

1.3 External tools and reproducibility

> **Construction validates the selected backends**
> The default evaluation order includes AlphaLoop, MATAD, FMFT, and pySecDec. Settings validation therefore probes FORM and pySecDec even if a later example only canonicalizes an expression. Vakint requires FORM 4.2.1 or newer and pySecDec 1.6.4 or newer. For matching without evaluation, use an empty evaluation order in Rust; in Python, provide an empty `evaluation_order` when constructing `Vakint`.

Analytic AlphaLoop, MATAD, and FMFT methods depend on FORM. The pySecDec method depends on both FORM and pySecDec and requires numerical masses and, where applicable, external momenta. Record tool versions, normalization convention, epsilon depth, decimal precision, backend order, and temporary-output reuse policy with any result intended to be reproduced.

1.4 Diagnostics and platform notes

Matching with an empty evaluation order needs neither FORM nor pySecDec. For backend evaluation, check the configured executable paths first. Verbose logging and retained temporary files are useful when an external tool fails:

```
VAKINT_NO_CLEAN_TMP_DIR=T RUST_LOG=DEBUG cargo run
```

On macOS, if a GNU-compiler link fails with missing `__emul...` symbols, retry the build with `EXTRA_MACOS_LIBS_FOR_GNU_GCC=T` set.

Vakint is used by GammaLoop for vacuum-integral work, but it owns its topology and evaluation conventions independently. Use this guide and the generated API/topology references for the selected version. Contributors can follow topology construction, canonicalization, backend selection, and external-process ownership in the Vakint implementation architecture.

2 Canonicalize your first Vakint integral

Vakint recognizes and canonicalizes single-scale vacuum integrals through both a Symbolica Python community module and a native Rust crate. Both routes below deliberately disable evaluation backends so the first success needs no FORM, MATAD, FMFT, or pySecDec installation.

Python
Use the community module for notebooks, exploratory matching, and Python algebra pipelines. The current Symbolica wheel bundles Vakint directly.

Use Vakint from Python →

Rust
Use the native crate when a Rust application owns integral expressions, matching policy, and later backend orchestration.

Use Vakint from Rust →

Continue with the matching and evaluation guide only after the canonical topology is understood and a backend policy is explicit.

3 Using Vakint from Python

Symbolica `2.2.0` bundles Vakint's community module. This matching-only workflow canonicalizes one loop with arbitrary input labels and invokes no external evaluation tool.

3.1 Install and verify the module

```
python -m venv .venv
. .venv/bin/activate
python -m pip install --upgrade pip
python -m pip install "symbolica==2.2.0"
python -c "import symbolica.community.vakint as vakint; print(vakint.__name__)"
```

There is no separate `vakint` Python wheel. Follow Symbolica's installation and license terms.

3.2 Canonicalize a one-loop integral

Save this as `vakint_quickstart.py`:

```python
from symbolica import E
from symbolica.community.vakint import Vakint

engine = Vakint(evaluation_order=[])
integral = E(
    "topo(prop(18,edge(7,7),k(99),muvsq,1))",
    default_namespace="vakint",
)
canonical = engine.to_canonical(integral, short_form=True)

assert "I1L" in str(canonical)
print(canonical)
```

Run `python vakint_quickstart.py`. The arbitrary propagator, edge, and momentum labels are normalized to Vakint's one-loop topology. An empty `evaluation_order` is intentional: it prevents this first use from probing FORM, MATAD, FMFT, or pySecDec.

> **Matching and evaluation are separate choices**
> Canonicalization answers which supported topology an expression represents. Tensor reduction and numerical or analytic evaluation add backend requirements and normalization choices. Enable them only after the canonical form is understood.

Use the Rust guide for the native matching API, or continue with the matching and evaluation guide before enabling a backend.

4 Using Vakint from Rust

This route parses and canonicalizes the same one-loop integral as the Python example while keeping matching policy and expression ownership in Rust.

> **Use the current Git API until the next crate release**
> The published `vakint 0.1.2` API differs from the current source while carrying the same version number. The pinned Git dependency below matches this manual. Switch to the next published Vakint version only after its Rustdoc exposes the `Vakint::new()` and explicit-settings API used here.

4.1 Create a Rust project

Use Rust 1.85 or newer:

```sh
cargo new vakint-quickstart
cd vakint-quickstart
cargo add vakint \
  --git https://github.com/alphal00p/gammaloop.git \
  --rev 41e0eddb39c7c668074af483ac1d566e46c247a8
cargo add symbolica@2.2.0 --no-default-features
```

Replace `src/main.rs` with:

```rust
use vakint::{
    vakint_parse, EvaluationOrder, Vakint, VakintExpression, VakintSettings,
};

fn main() -> Result<(), Box<dyn std::error::Error>> {
    let settings = VakintSettings {
        allow_unknown_integrals: false,
        evaluation_order: EvaluationOrder::empty(),
        ..VakintSettings::default()
    };
    let engine = Vakint::new()?;
    engine.validate_settings(&settings)?;

    let input = vakint_parse!(
        "topo(prop(18,edge(7,7),k(99),muvsq,1))"
    )?;
    let canonical = engine.to_canonical(&settings, input.as_view(), true)?;
    let canonical = VakintExpression::try_from(canonical)?;

    assert!(canonical.to_string().contains("I1L"));
    println!("{canonical}");
    Ok(())
}
```

Run `cargo run`. Success means Vakint recognizes the input as its canonical one-loop topology and exits without probing an evaluation executable. Symbolica's license terms apply to this native route as well as to the Python community module.

Continue with the matching and evaluation guide before adding tensor numerators or enabling analytic and numerical backends.

5 Tutorial

This tutorial parses and canonicalizes one vacuum integral in Rust. It deliberately disables all evaluation backends, so the first success exercises Vakint's topology library and momentum routing without requiring FORM, MATAD, FMFT, or pySecDec.

For the shortest installation and first-run path, begin with Using Vakint from Rust. Then return here to inspect topology matching and momentum routing in more detail.

5.1 Prerequisites

Use Rust 1.85 or newer and work from the GammaLoop source revision identified in this page's footer. The `latest` documentation follows that checkout's API; use an immutable documentation snapshot when working from a released package. Enter the repository development environment:

```
nix develop
cargo test --locked -p alphal00p-docs-examples
```

Vakint uses Symbolica; ensure your use satisfies its license terms. External integration tools are unnecessary for the matching-only settings below. The documentation test compiles this example against the same workspace revision as the published Rustdoc.

5.2 Match a one-loop topology

The following is a complete program. The documentation test compiles it directly; to inspect its output, place the same program in a scratch binary target in this checkout and run that target.

```rust
use vakint::{
    vakint_parse, EvaluationOrder, Vakint, VakintExpression, VakintSettings,
};

fn main() -> Result<(), Box<dyn std::error::Error>> {
    let settings = VakintSettings {
        allow_unknown_integrals: false,
        evaluation_order: EvaluationOrder::empty(),
        ..VakintSettings::default()
    };
    let vakint = Vakint::new()?;
    vakint.validate_settings(&settings)?;

    let input = vakint_parse!(
        "topo(prop(1,edge(1,1),k(1),muvsq,1))"
    )?;
    let canonical = vakint.to_canonical(&settings, input.as_view(), true)?;

    println!("input: {}", VakintExpression::try_from(input)?);
    println!("canonical: {}", VakintExpression::try_from(canonical)?);
    Ok(())
}
```

Runtime success means Vakint recognizes the propagator as a supported one-loop vacuum topology, the `canonical:` line contains the short form `topo(I1L(muvsq,1))`, and the program exits without probing an external executable. Other canonical spellings may reorder identifiers or momentum routing; the normalized result, not the input's labels, is the comparison boundary.

Verification scope and cost
The docs harness compiles this source against the current workspace and syntax-checks the setup command. Run the program in a provisioned checkout to verify that both printed forms are produced and the canonical form is recognized. Expect a dependency-heavy cold build to take minutes and the matching operation itself to take seconds. Symbolica may refuse a second concurrent unlicensed process; stop the other instance or use a licensed environment before retrying.

An empty evaluation order is intentional
`EvaluationOrder::empty()` makes this a topology-matching tutorial. Default settings include analytic and numerical backends. Validating the explicit empty order before `to_canonical` prevents an accidental external-backend probe in a matching-only workflow.

5.3 Move from matching to evaluation

Once the canonical form is understood, choose one supported backend explicitly, validate its dependencies, then apply the stages in order: tensor reduction, integral evaluation, and numerical substitution. Record the normalization factor, epsilon depth, decimal precision, backend settings, and tool versions with the result. Reuse one `Vakint` engine because topology construction has a cost.

5.4 Troubleshooting and next steps

- An unrecognized-integral error means the propagators, powers, masses, or momentum routing did not match a registered topology and `allow_unknown_integrals` is false. Print the long form and compare it with the supported-topology reference.
- A FORM or pySecDec version error means a nonempty evaluation order slipped into the settings; return to `EvaluationOrder::empty()` for pure canonicalization.

- A Symbolica concurrency error means another unlicensed Symbolica process is active; stop it or retry in an environment licensed for concurrent instances.
- If two inputs look different after matching, compare their canonical forms rather than their original propagator numbering.
- Continue with the topology-and-evaluation guide before enabling a backend or adding tensor numerators.

6 Topology matching, reduction, and evaluation

Vakint recognizes an integral only after its propagators, masses, powers, and momentum routing can be mapped to a supported topology pattern. The topology reference lists the patterns available in the selected version.

6.1 Parsing and topology matching

A `VakintExpression` is a sum of numerator/topology pairs. Parsing checks the Vakint namespace; matching then searches allowed substitutions and momentum shifts. Unknown-topology handling is a setting: preserving an unknown term is useful for partial simplification, but does not make that term evaluable.

Canonical long form is explicit and suitable for auditing. Canonical short form is convenient for display and library lookup. Both represent the same matched topology only after canonicalization has succeeded.

6.2 Normalization and canonicalization

Normalization conventions cover loop-measure factors, mass scales, epsilon powers, and the requested expansion depth. Set them before comparing results from different backends. Momentum canonicalization may rename loop variables and reorder propagators; compare canonical expressions rather than input spelling when testing equivalence.

> **Matching does not require an evaluation backend**
> Parsing, matching, normalization, and canonical routing work without FORM or pySecDec when the evaluation order is empty. Use this configuration when you only need a canonical topology.

6.3 Tensor reduction

Tensor numerators are reduced to scalar integrals before analytic evaluation where required. Reduction depends on the topology, Lorentz rank, dimension convention, and scalar-product normalization. Tensor reduction requires FORM. When diagnosing a mismatch, keep the FORM input and Vakint temporary directory so that the failing reduction can be inspected.

6.4 Evaluation order and backends

The evaluation order is an ordered selection policy, not a request to combine backend results. Vakint skips methods whose capability declaration does not match the canonical topology, then uses the first matching method. An error from that selected backend is returned immediately; it does not trigger fallback to the next method. AlphaLoop, MATAD, and FMFT are analytic FORM-backed paths. pySecDec is numerical and additionally needs numeric parameters and external momenta where the topology requires them.

For reproducible comparisons record:

- the canonical topology and normalization convention;
- epsilon expansion depth and decimal precision;
- the ordered backend list and backend-specific settings;
- FORM, pySecDec, MATAD, and FMFT versions where applicable;
- parameter substitutions and any retained temporary-output directory.

6.5 Configure one analytic tensor reduction

This program makes normalization, precision, and backend order explicit before reducing a rank-two one-loop numerator. It compiles without running external tools in the documentation harness; running it requires a supported FORM installation for the AlphaLoop path.

```rust
use std::collections::HashMap;

use vakint::{
    EvaluationOrder, LoopNormalizationFactor, Vakint, VakintSettings, vakint_parse,
};

fn main() -> Result<(), Box<dyn std::error::Error>> {
    let settings = VakintSettings {
        allow_unknown_integrals: false,
        integral_normalization_factor: LoopNormalizationFactor::MSbar,
        number_of_terms_in_epsilon_expansion: 2,
        run_time_decimal_precision: 32,
        evaluation_order: EvaluationOrder::alphaloop_only(),
        ..VakintSettings::default()
    };
    let vakint = Vakint::new()?;
    let input = vakint_parse!(
        "(k(1,1)*k(1,2)+k(1,3)*p(1,3))*topo(prop(1,edge(1,1),k(1),muvsq,1))"
    )?;
    let canonical = vakint.to_canonical(&settings, input.as_view(), true)?;
    let scalar = vakint.tensor_reduce(&settings, canonical.as_view())?;
    let evaluated = vakint.evaluate_integral(&settings, scalar.as_view())?;

    let parameter_values = HashMap::from([
        ("muvsq".to_owned(), 1.0),
        (settings.mu_r_sq_symbol.clone(), 1.0),
    ]);
    let real_parameters = vakint.params_from_f64(&settings, &parameter_values);
    let complex_parameters = HashMap::default();
    let (numerical, numerical_error) = vakint.numerical_evaluation(
        &settings,
        evaluated.as_view(),
        &real_parameters,
        &complex_parameters,
        None,
    )?;

    for (epsilon_power, coefficient) in numerical.get_epsilon_coefficients() {
        println!("epsilon^{epsilon_power}: {coefficient}");
    }
    if let Some(error) = numerical_error {
        println!("backend uncertainty: {error}");
    }
    Ok(())
}
```

The result invariant is an MS-bar Laurent series in the dimensional regulator: the reduced expression no longer contains the original loop-momentum numerator, and only the first successful method in `alphaloop_only()` contributes. Inspect the exact `VakintSettings`, tensor reduction, and evaluation Rustdoc before changing normalization or backend order.

6.6 Numerical substitution and result interpretation

Backend evaluation returns a symbolic Laurent series. Numerical substitution is a separate boundary: supply every remaining real/complex parameter and, when the numerator contains external scalar products, the external

four-momenta. `params_from_f64` and `externals_from_f64` convert ordinary inputs to the configured decimal precision before the series is evaluated.

`NumericalEvaluationResult` stores sorted `(epsilon power, complex coefficient)` pairs. Negative powers are poles, power zero is the finite term, and positive powers are higher orders. The optional second result carries a backend-reported uncertainty series when the evaluated expression contains one. Compare corresponding powers; do not collapse the series to one complex number or silently discard the uncertainty.

Use `partial_numerical_evaluation` when you intentionally want to substitute known constants and retain unresolved symbolic factors for inspection. Use `numerical_evaluation` for the final boundary: it reports an error when tensor structure or a required symbol remains. This distinction prevents a partially substituted expression from being mistaken for a fully numerical result.

> **Interpret failures at the owning stage**
> An engine-construction error means the topology library could not be initialized. A canonicalization error means the propagators did not match a supported topology. A reduction error points to the numerator, Lorentz rank, or FORM translation; retain the temporary directory in that case. An evaluation error after successful reduction means settings validation, the selected backend's capabilities, or the backend invocation failed.

> **Python is an embedded community module**
> `symbolica.community.vakint` is registered into a Symbolica installation; it is not a standalone PyPI package. Constructing its `Vakint` class validates the configured backends. Pass an empty evaluation order for pure matching work on machines without FORM/pySecDec.

See the supported-topology and external-dependency reference for the patterns and minimum tool versions available here. Their Rust definitions begin in Vakint's topology module.

6.7 Methods and software to cite

Vakint combines distinct methods rather than treating every backend as interchangeable. Cite the software version and the method actually selected for the reported result:

- FORM for the symbolic reduction engine;
- MATAD when its massive tadpole tables are used;
- FMFT for the four-loop fully massive tadpole path; and
- pySecDec for numerical sector decomposition.

Vakint also uses Symbolica for expression manipulation. Record the Vakint revision, normalization, epsilon depth, precision, selected backend and dependency versions with the result; a generic citation to the package does not encode those choices.

7 Rust and Python APIs

> **API reference**
> The reference for this version lists public items, signatures, fields, defaults, feature requirements, and links to their source definitions.
>
> Rust packages: `vakint`
> Python modules: `symbolica.community.vakint`

7.1 Rust package

The main Rust types make the workflow state explicit:

- `Vakint` owns the topology library and matching/evaluation operations;

- `VakintSettings` owns symbols, precision, normalization, tool paths, validation, and backend order;
- `VakintExpression` and `VakintTerm` separate numerators from topology atoms;
- `EvaluationOrder` and `EvaluationMethod` select AlphaLoop, MATAD, FMFT, or pySecDec;
- `NumericalEvaluationResult` stores Laurent coefficients in the dimensional regulator.

For a dependency-free canonicalization setup, select an empty backend order before validating:

```
use vakint::{EvaluationOrder, Vakint, VakintSettings};

let engine = Vakint::new()?;
let settings = VakintSettings {
    evaluation_order: EvaluationOrder::empty(),
    ..VakintSettings::default()
};
engine.validate_settings(&settings)?;
```

`vakint_parse!` and the related helpers parse Symbolica expressions in Vakint's namespace. Handle parse failures before passing a user-supplied expression to matching or evaluation.

The Rust orientation identifies the compiled crates, versions, and feature matrix and links directly to native Rustdoc. The generated topology reference records the runtime topology library and supported external-tool versions for this release.

7.2 Python community module

Python availability
Python users import `symbolica.community.vakint`; Vakint is not distributed as a standalone Python package. Check that the installed Symbolica distribution includes the Vakint community module before relying on this interface.

Install and verify the published assembly with `python -m pip install --upgrade symbolica` and `python -c "import symbolica.community.vakint"`. Custom Symbolica builds can add Vakint to the symbolica-community assembly, enable the `symbolica_community_module` feature, and register `VakintWrapper` while building the extension.

The structured Python reference covers the four exported classes: `Vakint`, `VakintExpression`, `VakintEvaluationMethod`, and `VakintNumericalResult`. This example constructs a matching-only engine without probing external backends:

```
from symbolica import E
from symbolica.community.vakint import Vakint

vakint = Vakint(evaluation_order=[])
expr = E(
    "topo(prop(18,edge(7,7),k(99),muvsq,1))",
    default_namespace="vakint",
)
canonical = vakint.to_canonical(expr, short_form=True)
assert "I1L" in str(canonical)
```

`Vakint` exposes canonicalization, tensor reduction, integral evaluation, complete evaluation, and numerical conversion. `VakintEvaluationMethod` constructs configured backend choices. `VakintNumericalResult` converts Laurent coefficients to Python tuples and compares results with thresholds and optional errors.

Construct one engine and reuse it. When enabling pySecDec, provide numerical masses and external momenta where the topology requires them. Reuse existing pySecDec output only while debugging, and remove it before evaluating changed inputs.

Source starting points are the Rust API and the Python module.

8 Version history

> **History coverage**
> Vakint does not yet have a standalone changelog. The package metadata, tagged repository history, and API reference are the available version record.

8.1 Components and version sources

- Rust package: vakint package metadata
- Release record: Vakint repository tags
- Usage and external tools: matching and evaluation guide

The Rust crate and the Python interface included with Symbolica can be released on different schedules. Record both versions when sharing a calculation or reporting a problem.

8.2 Upgrade checklist

Pin the Vakint version together with FORM, pySecDec, MATAD, and FMFT versions used for a calculation. Then compare:

- supported topologies and canonical momentum routing;
- normalization conventions and epsilon-expansion defaults;
- tensor reduction and analytic backend behavior;
- pySecDec integration and numerical precision;
- Rust and Python method signatures and feature requirements.

8.3 Reproducibility record

Record the Rust crate, Symbolica/Python module, FORM, pySecDec, MATAD, and FMFT versions together with the selected topology and normalization settings.

Python API reference

The supported Python packages available in this version are listed below. The HTML manual provides a separate page for every public class and function; this printable edition keeps a compact inventory. Rust APIs use canonical Rustdoc in the HTML manual.

`vakint-community`

Exports registered by vakint.

`Vakint`

Vakint engine and settings used for matching, reduction, and evaluation.

Vakint engine and settings used for matching, reduction, and evaluation.

Construct one instance and reuse it: initialization processes the complete topology library.

class Vakint:

Requires features: `symbolica_community_module, python_stubgen`

`__new__`

Kind: `AssociatedFunction`

```
def __new__(cls, run_time_decimal_precision: Optional[int] = None, evaluation_order:
Optional[Sequence[VakintEvaluationMethod]] = None, epsilon_symbol:
Optional[Expression] = None, mu_r_sq_symbol: Optional[Expression] = None,
form_exe_path: Optional[str] = None, python_exe_path: Optional[str] = None,
verify_numerator_identification: Optional[bool] = None,
integral_normalization_factor: Optional[str] = None, allow_unknown_integrals:
Optional[bool] = None, clean_tmp_dir: Optional[bool] = None,
number_of_terms_in_epsilon_expansion: Optional[int] = None, use_dot_product_notation:
Optional[bool] = None, temporary_directory: Optional[str] = None) -> Vakint:
```

Create a new Vakint instance, specifying details of the evaluation stack. Note that the same instance can be recycled across multiple evaluations.
Note that the creation of a Vakint instance involves the processing and creation of the library of all known topologies, which can be time consuming.

Examples
```python
>>> from symbolica.community.vakint import Vakint
>>> vakint = Vakint(evaluation_order=[])
>>> vakint is not None
True
```

An empty evaluation order is appropriate for matching, canonicalization, and tensor reduction. Add explicit `VakintEvaluationMethod` entries before evaluating an integral;
construction validates the executables required by those entries.

Parameters
----

run_time_decimal_precision : Optional[int]
    The decimal precision to be used during the evaluation. Default is 17.
evaluation_order : Optional[Sequence[VakintEvaluationMethod]]
    A list of `VakintEvaluationMethod` instances specifying the order in which

evaluation methods are to be applied. Default is all available methods in a sensible order.
epsilon_symbol : Optional[Expression]
    The symbol to be used for the dimensional regularisation parameter epsilon. Default is "ε".
mu_r_sq_symbol : Optional[Expression]
    The symbol to be used for the renormalisation scale squared. Default is "mursq".
form_exe_path : Optional[str]
    The path to the FORM executable. Default is "form".
python_exe_path : Optional[str]
    The path to the Python executable. Default is "python3".
verify_numerator_identification : Optional[bool]
    Whether to verify the identification of numerator structures. Default is True.
integral_normalization_factor : Optional[str]
    The normalization factor to be used for integrals. Can be "MSbar", "pySecDec", "FMFTandMATAD" or a custom string. Default is "MSbar".
allow_unknown_integrals : Optional[bool]
    Whether to allow unknown integrals to be processed. Default is True.
clean_tmp_dir : Optional[bool]
    Whether to clean the temporary directory after evaluation. Default is True, unless the environment variable VAKINT_NO_CLEAN_TMP_DIR is set.
number_of_terms_in_epsilon_expansion : Optional[int]
    The number of terms in the epsilon expansion to be computed. Default is 4.
use_dot_product_notation : Optional[bool]
    Whether to use dot product notation for scalar products. Default is False.
temporary_directory : Optional[str]
    The path to the temporary directory to be used. Default is None, in which case a system temporary directory will be used.

run_time_decimal_precision

Kind: Parameter

Optional[int]

Default: None

The decimal precision to be used during the evaluation. Default is 17.

evaluation_order

Kind: Parameter

Optional[Sequence[VakintEvaluationMethod]]

Default: None

A list of `VakintEvaluationMethod` instances specifying the order in which evaluation methods are to be applied. Default is all available methods in a sensible order.

epsilon_symbol

Kind: Parameter

Optional[Expression]

Default: None

The symbol to be used for the dimensional regularisation parameter epsilon. Default is "ε".

mu_r_sq_symbol

Kind: Parameter

Optional[Expression]

Default: None

The symbol to be used for the renormalisation scale squared. Default is "mursq".

form_exe_path
Kind: Parameter

Optional[str]

Default: None

The path to the FORM executable. Default is "form".

python_exe_path
Kind: Parameter

Optional[str]

Default: None

The path to the Python executable. Default is "python3".

verify_numerator_identification
Kind: Parameter

Optional[bool]

Default: None

Whether to verify the identification of numerator structures. Default is True.

integral_normalization_factor
Kind: Parameter

Optional[str]

Default: None

The normalization factor to be used for integrals. Can be "MSbar", "pySecDec", "FMFTandMATAD" or a custom string. Default is "MSbar".

allow_unknown_integrals
Kind: Parameter

Optional[bool]

Default: None

Whether to allow unknown integrals to be processed. Default is True.

clean_tmp_dir
Kind: Parameter

Optional[bool]

Default: None

Whether to clean the temporary directory after evaluation. Default is True, unless the environment variable VAKINT_NO_CLEAN_TMP_DIR is set.

number_of_terms_in_epsilon_expansion
Kind: Parameter

Optional[int]

Default: None

The number of terms in the epsilon expansion to be computed. Default is 4.

use_dot_product_notation

Kind: Parameter

Optional[bool]

Default: None

Whether to use dot product notation for scalar products. Default is False.

temporary_directory

Kind: Parameter

Optional[str]

Default: None

The path to the temporary directory to be used. Default is None, in which case a system temporary directory will be used.

numerical_result_from_expression

Kind: Method

def numerical_result_from_expression(self, expr: Expression) -> VakintNumericalResult:

Interpret a Symbolica expression as a numerical Laurent series in epsilon.

Examples
```python
>>> from symbolica import E
>>> from symbolica.community.vakint import Vakint
>>> vakint = Vakint(evaluation_order=[])
>>> result = vakint.numerical_result_from_expression(
...     E("vakint::ε^-2 + 1 + 0.12*vakint::ε^-1")
... )
>>> sorted(exponent for exponent, _ in result.to_list())
[-2, -1, 0]
```

Parameters
----

expr : Expression
  A Symbolica expression representing a Laurent series in the dimensional regularisation parameter epsilon specified in the vakint engine.

expr

Kind: Parameter

Expression

A Symbolica expression representing a Laurent series in the dimensional regularisation parameter epsilon specified in the vakint engine.

numerical_evaluation

Kind: Method

```
def numerical_evaluation(self, evaluated_integral: Any, params: Mapping[str, float],
externals: Optional[Mapping[int, tuple[float, float, float, float]]] = None) ->
tuple[VakintNumericalResult, Optional[VakintNumericalResult]]:
```

Substitute numerical parameters into an integral already evaluated parametrically by Vakint.

Examples
```python
>>> from symbolica import E
>>> from symbolica.community.vakint import Vakint
>>> vakint = Vakint(evaluation_order=[])
>>> evaluated = E(
...     "muvsq*vakint::ε^-1 + mursq",
...     default_namespace="vakint",
... )
>>> result, error = vakint.numerical_evaluation(
...     evaluated,
...     {"muvsq": 2.0, "mursq": 3.0},
... )
>>> sorted(exponent for exponent, _ in result.to_list())
[-1, 0]
>>> error is None
True
```

Parameters
----

evaluated_integral : Expression
  A Symbolica expression representing an integral that has been evaluated parametrically by Vakint.
params : Dict[str, float]
  A dictionary mapping parameter names to their numerical values.
externals : Optional[Dict[int, Tuple[float, float, float, float]]]
  An optional dictionary mapping external momentum indices to their numerical 4-vector values.

evaluated_integral

Kind: Parameter

Any

A Symbolica expression representing an integral that has been evaluated parametrically by Vakint.

params

Kind: Parameter

Mapping[str, float]

A dictionary mapping parameter names to their numerical values.

externals

Kind: Parameter

Optional[Mapping[int, tuple[float, float, float, float]]]

Default: None

An optional dictionary mapping external momentum indices to their numerical 4-vector values.

numerical_result_to_expression

Kind: Method

def numerical_result_to_expression(self, result: VakintNumericalResult) -> Expression:

Convert a Vakint numerical result to a Symbolica Laurent-series expression.

Examples
```python
>>> from symbolica.community.vakint import Vakint, VakintNumericalResult
>>> vakint = Vakint(evaluation_order=[])
>>> result = VakintNumericalResult([
...     (-1, (2.0, 0.0)),
...     (0, (3.0, 0.0)),
... ])
>>> expression = vakint.numerical_result_to_expression(result)
>>> "ε" in str(expression)
True
```

result

Kind: Parameter

VakintNumericalResult

to_canonical

Kind: Method

def to_canonical(self, integral_expression: Expression, short_form: Optional[bool] = None) -> Expression:

Convert a Vakint expression to canonical momentum routing and topology numbering.

Examples
```python
>>> from symbolica import E
>>> from symbolica.community.vakint import Vakint
>>> vakint = Vakint(evaluation_order=[])
>>> integral = E(
...     "topo(prop(18,edge(7,7),k(99),muvsq,1))",
...     default_namespace="vakint",
... )
>>> canonical = vakint.to_canonical(integral, short_form=True)
>>> "I1L" in str(canonical)
True
```

Parameters
---

integral_expression : Expression

```
  A Symbolica expression representing a vakint integral.
short_form : Optional[bool]
  Whether to use the short form for the topology representation. Default is False.
```

integral_expression

Kind: Parameter

Expression

A Symbolica expression representing a vakint integral.

short_form

Kind: Parameter

Optional[bool]

Default: None

Whether to use the short form for the topology representation. Default is False.

tensor_reduce

Kind: Method

```
def tensor_reduce(self, integral_expression: Expression) -> Expression:
```

Reduce the tensor integrals in a Vakint expression to scalar integrals.

```
## Examples
```python
>>> from symbolica import E
>>> from symbolica.community.vakint import Vakint
>>> vakint = Vakint(evaluation_order=[])
>>> integral = E(
...     "k(1,101)*k(1,102)*topo(prop(1,edge(1,1),k(1),muvsq,1))",
...     default_namespace="vakint",
... )
>>> reduced = vakint.tensor_reduce(integral)
>>> "g(101,102)" in str(reduced)
True
```

Parameters
----
integral_expression : Expression
   A Symbolica expression representing a vakint integral.
```

integral_expression

Kind: Parameter

Expression

A Symbolica expression representing a vakint integral.

evaluate_integral

Kind: Method

```
def evaluate_integral(self, integral_expression: Expression) -> Expression:
```

Perform the parametric evaluation of *only the integral* appearing in the Symbolica
expression given in input representing a vakint integral.
The numerator is left unchanged.

Examples
```python
>>> from symbolica import E
>>> from symbolica.community.vakint import Vakint, VakintEvaluationMethod
>>> vakint = Vakint(
...     evaluation_order=[VakintEvaluationMethod.new_alphaloop_method()]
... )
>>> integral = E(
...     "topo(prop(1,edge(1,1),k(1),muvsq,1))",
...     default_namespace="vakint",
... )
>>> evaluated = vakint.evaluate_integral(integral)
>>> "ε" in str(evaluated)
True
```

The AlphaLoop evaluation method used here invokes FORM. Configure `form_exe_path` if FORM is not available as `form` on `PATH`.

Parameters
----

integral_expression : Expression
  A Symbolica expression representing a vakint integral.

integral_expression

Kind: Parameter

Expression

A Symbolica expression representing a vakint integral.

evaluate

Kind: Method

def evaluate(self, integral_expression: Expression) -> Expression:

Perform the complete parametric evaluation of the Vakint integral represented by the Symbolica expression given in input.
Note that the tensor reduction will be automatically performed on the input given.

Examples
```python
>>> from symbolica import E
>>> from symbolica.community.vakint import Vakint, VakintEvaluationMethod
>>> vakint = Vakint(
...     evaluation_order=[VakintEvaluationMethod.new_alphaloop_method()]
... )
>>> integral = E(
...     "k(1,101)*k(1,102)*topo(prop(1,edge(1,1),k(1),muvsq,1))",
...     default_namespace="vakint",
... )
>>> evaluated = vakint.evaluate(integral)
>>> "g(101,102)" in str(evaluated)
True
```

```
This complete path performs tensor reduction before integral evaluation and therefore
has the same FORM requirement as `evaluate_integral` for the AlphaLoop method.

Parameters
----------

integral_expression : Expression
  A Symbolica expression representing a vakint integral.
```

integral_expression

Kind: Parameter

```
Expression
```

```
A Symbolica expression representing a vakint integral.
```

Exhaustive reference · Source

VakintEvaluationMethod

```
One configured backend in a `Vakint` instance's evaluation order.
```

```
One configured backend in a `Vakint` instance's evaluation order.
```

```
class VakintEvaluationMethod:
```

Requires features: `symbolica_community_module, python_stubgen`

__str__

Kind: Method

```
def __str__(self) -> str:
```

```
String representation of the evaluation method.
```

new_alphaloop_method

Kind: AssociatedFunction

```
def new_alphaloop_method(cls) -> VakintEvaluationMethod:
```

```
Create a new VakintEvaluationMethod instance representing the AlphaLoop method.
This method does not take any parameters.

## Examples
```python
>>> from symbolica.community.vakint import VakintEvaluationMethod
>>> alphaloop_method = VakintEvaluationMethod.new_alphaloop_method()
>>> "AlphaLoop" in str(alphaloop_method)
True
```
```

new_matad_method

Kind: AssociatedFunction

```
def new_matad_method(cls, expand_masters: Optional[bool] = None, susbstitute_masters:
Optional[bool] = None, substitute_hpls: Optional[bool] = None,
direct_numerical_substition: Optional[bool] = None) -> VakintEvaluationMethod:
```

```
Create a new VakintEvaluationMethod instance representing the MATAD method.

## Examples
```python
>>> from symbolica.community.vakint import VakintEvaluationMethod
```
```

```
>>> matad_method = VakintEvaluationMethod.new_matad_method(
...     expand_masters=True,
...     susbstitute_masters=True,
...     substitute_hpls=True,
...     direct_numerical_substition=True,
... )
>>> "MATAD" in str(matad_method)
True
```

Parameters
----

expand_masters : Optional[bool]
   Whether to expand master integrals. Default is True.
susbstitute_masters : Optional[bool]
   Whether to substitute master integrals. Default is True.
substitute_hpls : Optional[bool]
   Whether to substitute harmonic polylogarithms. Default is True.
direct_numerical_substition : Optional[bool]
   Whether to perform direct numerical substitution. Default is True.

Notes

`susbstitute_masters` and `direct_numerical_substition` retain their historical misspellings for API compatibility. They mean `substitute_masters` and `direct_numerical_substitution`, respectively.

expand_masters

Kind: Parameter

Optional[bool]

Default: None

Whether to expand master integrals. Default is True.

susbstitute_masters

Kind: Parameter

Optional[bool]

Default: None

Whether to substitute master integrals. Default is True.

substitute_hpls

Kind: Parameter

Optional[bool]

Default: None

Whether to substitute harmonic polylogarithms. Default is True.

direct_numerical_substition

Kind: Parameter

Optional[bool]

Default: None

Whether to perform direct numerical substitution. Default is True.

new_fmft_method

Kind: AssociatedFunction

def new_fmft_method(cls, expand_masters: Optional[bool] = None, susbstitute_masters: Optional[bool] = None) -> VakintEvaluationMethod:

Create a new VakintEvaluationMethod instance representing the FMFT method.

Examples
```python
>>> from symbolica.community.vakint import VakintEvaluationMethod
>>> fmft_method = VakintEvaluationMethod.new_fmft_method(
...     expand_masters=True,
...     susbstitute_masters=True,
... )
>>> "FMFT" in str(fmft_method)
True
```

Parameters
----

expand_masters : Optional[bool]
  Whether to expand master integrals. Default is True.
susbstitute_masters : Optional[bool]
  Whether to substitute master integrals. Default is True.

Notes

`susbstitute_masters` retains its historical misspelling for API compatibility;
it means `substitute_masters`.

expand_masters

Kind: Parameter

Optional[bool]

Default: None

Whether to expand master integrals. Default is True.

susbstitute_masters

Kind: Parameter

Optional[bool]

Default: None

Whether to substitute master integrals. Default is True.

new_pysecdec_method

Kind: AssociatedFunction

def new_pysecdec_method(cls, quiet: Optional[bool] = None, relative_precision: Optional[float] = None, min_n_evals: Optional[int] = None, max_n_evals: Optional[int] = None, reuse_existing_output: Optional[str] = None, numerical_masses:

```
Optional[Mapping[str, float]] = None, numerical_external_momenta:
Optional[Mapping[int, tuple[float, float, float, float]]] = None) ->
VakintEvaluationMethod:
```

Create a new VakintEvaluationMethod instance representing the numerical pySecDec method.

Examples

```python
>>> from symbolica.community.vakint import VakintEvaluationMethod
>>> pysecdec_method = VakintEvaluationMethod.new_pysecdec_method(
...     quiet=True,
...     relative_precision=1e-7,
...     min_n_evals=10_000,
...     max_n_evals=1_000_000_000_000,
...     reuse_existing_output=None,
...     numerical_masses={"muvsq": 1.0},
...     numerical_external_momenta={
...         1: (1.0, 0.0, 0.0, 0.0),
...         2: (0.0, 1.0, 0.0, 0.0),
...     },
... )
>>> "PySecDec" in str(pysecdec_method)
True
```

pySecDec performs numerical evaluation, so every required mass and external momentum must have a numerical value. Constructing this method does not run pySecDec; evaluation
requires a working Python/pySecDec installation.

Parameters
---

quiet : Optional[bool]
   Whether to suppress output from pySecDec. Default is True.
relative_precision : Optional[float]
   The relative precision to be achieved in the numerical integration. Default is 1e-7.
min_n_evals : Optional[int]
   The minimum number of evaluations to be performed in the numerical integration. Default is 10,000.
max_n_evals : Optional[int]
   The maximum number of evaluations to be performed in the numerical integration. Default is 1,000,000,000,000.
reuse_existing_output : Optional[str]
   Path to existing pySecDec output to reuse. Default is None.
numerical_masses : Optional[Dict[str, float]]
   A dictionary mapping mass parameter names to their numerical values. Default is an empty dictionary.
numerical_external_momenta : Optional[Dict[int, Tuple[float, float, float, float]]]
   A dictionary mapping external momentum indices to their numerical 4-vector values. Default is an empty dictionary.

quiet

Kind: Parameter

Optional[bool]

Default: None

Whether to suppress output from pySecDec. Default is True.

relative_precision

Kind: Parameter

Optional[float]

Default: None

The relative precision to be achieved in the numerical integration. Default is 1e-7.

min_n_evals

Kind: Parameter

Optional[int]

Default: None

The minimum number of evaluations to be performed in the numerical integration. Default is 10,000.

max_n_evals

Kind: Parameter

Optional[int]

Default: None

The maximum number of evaluations to be performed in the numerical integration. Default is 1,000,000,000,000.

reuse_existing_output

Kind: Parameter

Optional[str]

Default: None

Path to existing pySecDec output to reuse. Default is None.

numerical_masses

Kind: Parameter

Optional[Mapping[str, float]]

Default: None

A dictionary mapping mass parameter names to their numerical values. Default is an empty dictionary.

numerical_external_momenta

Kind: Parameter

Optional[Mapping[int, tuple[float, float, float, float]]]

Default: None

A dictionary mapping external momentum indices to their numerical 4-vector values. Default is an empty dictionary.

Exhaustive reference · Source

VakintExpression

A Vakint integral split into its numerator and normalized topology structure.

A Vakint integral split into its numerator and normalized topology structure.

Construct this wrapper from a Symbolica expression before applying Vakint operations.

```
class VakintExpression:
```

Requires features: `symbolica_community_module, python_stubgen`

__str__

Kind: Method

```
def __str__(self) -> str:
```

String representation of the VakintExpression.

to_expression

Kind: Method

```
def to_expression(self) -> Expression:
```

Convert the VakintExpression back to a Symbolica Expression.

Examples

```python
>>> from symbolica import E
>>> from symbolica.community.vakint import VakintExpression
>>> integral = VakintExpression(E('''
...     k(1,11)*k(1,11)
...     *topo(prop(1,edge(1,1),k(1),muvsq,1))
... ''', default_namespace="vakint"))
>>> "topo(" in str(integral.to_expression())
True
```

__new__

Kind: AssociatedFunction

```
def __new__(cls, atom: Any) -> VakintExpression:
```

Split a Symbolica expression into Vakint numerator and topology components.

Examples

```python
>>> from symbolica import E
>>> from symbolica.community.vakint import VakintExpression
>>> integral = E('''
...     (
...         k(1,11)*k(2,11)*k(1,22)*k(2,22)
...       + p(1,11)*k(3,11)*k(3,22)*p(2,22)
...       + p(1,11)*p(2,11)*(k(2,22)+k(1,22))*k(2,22)
...     )*topo(
...          prop(1,edge(1,2),k(1),muvsq,1)
...         * prop(2,edge(2,3),k(2),muvsq,1)
...         * prop(3,edge(3,1),k(3),muvsq,1)
...         * prop(4,edge(1,4),k(3)-k(1),muvsq,1)
...         * prop(5,edge(2,4),k(1)-k(2),muvsq,1)
```

```
...         * prop(6,edge(3,4),k(2)-k(3),muvsq,1)
...     )
... ''', default_namespace="vakint")
>>> wrapped = VakintExpression(integral)
>>> "topo(" in str(wrapped)
True
```

Parameters
----

atom : Expression
  A Symbolica Expression containing a vakint integral, i.e. a sum of terms, each a product of a numerator and a `vakint::topo(...)` structure.

atom

Kind: Parameter

Any

A Symbolica Expression containing a vakint integral, i.e. a sum of terms, each a product of a numerator and a `vakint::topo(...)` structure.

Exhaustive reference · Source

VakintNumericalResult

Numerical Laurent series in the dimensional-regularization parameter epsilon.

Numerical Laurent series in the dimensional-regularization parameter epsilon.

class VakintNumericalResult:

Requires features: symbolica_community_module, python_stubgen

__str__

Kind: Method

def __str__(self) -> str:

String representation of the numerical result.

Examples
```python
>>> from symbolica.community.vakint import VakintNumericalResult
>>> result = VakintNumericalResult([
...     (-3, (0.0, -11440.53140354612)),
...     (-2, (0.0, 57169.95521898031)),
...     (-1, (0.0, -178748.9838377694)),
...     (0, (0.0, 321554.1122184795)),
... ])
>>> str(result)
ε^-3 : (0+-11440.5314035461i)
ε^-2 : (0+57169.9552189803i)
ε^-1 : (0+-178748.983837769i)
ε^ 0 : (0+321554.112218480i)
```

to_list

Kind: Method

def to_list(self) -> list[tuple[int, tuple[float, float]]]:

Convert the numerical result to a native Python list of (epsilon exponent, (real, imag)) tuples.

Examples
```python
>>> from symbolica.community.vakint import VakintNumericalResult
>>> result = VakintNumericalResult([
...     (-3, (0.0, -11440.53140354612)),
...     (-2, (0.0, 57169.95521898031)),
...     (-1, (0.0, -178748.9838377694)),
...     (0, (0.0, 321554.1122184795)),
... ])

>>> values = result.to_list()
>>> values[0]
(-3, (0.0, -11440.53140354612))
```

__new__

Kind: AssociatedFunction

def __new__(cls, values: Sequence[tuple[int, tuple[float, float]]]) -> VakintNumericalResult:

Create a numerical Laurent series from `(epsilon exponent, (real, imaginary))` tuples.

Examples
```python
>>> from symbolica.community.vakint import VakintNumericalResult
>>> result = VakintNumericalResult([
...     (-3, (0.0, -11440.53140354612)),
...     (-2, (0.0, 57169.95521898031)),
...     (-1, (0.0, -178748.9838377694)),
...     (0, (0.0, 321554.1122184795)),
... ])
>>> len(result.to_list())
4
```

Parameters
----

values : List[Tuple[int, Tuple[float, float]]]
   A list of tuples, each containing an integer exponent of epsilon and a tuple of two floats
   representing the real and imaginary parts of the coefficient.

values

Kind: Parameter

Sequence[tuple[int, tuple[float, float]]]

A list of tuples, each containing an integer exponent of epsilon and a tuple of two floats representing the real and imaginary parts of the coefficient.

compare_to

Kind: Method

def compare_to(self, other: VakintNumericalResult, relative_threshold: float, error: Optional[VakintNumericalResult] = None, max_pull: Optional[float] = None) -> tuple[bool, str]:

Compare this numerical result to another, returning a tuple of (bool, str) where the bool indicates whether the results match within the specified thresholds,
and the str provides details of the comparison.

Examples
```python
>>> from symbolica.community.vakint import VakintNumericalResult
>>> result1 = VakintNumericalResult([
...     (-3, (0.0, -11440.53140354612)),
... ])
>>> result2 = VakintNumericalResult([
...     (-3, (0.0, -11440.53140354612)),
...     (-2, (0.0, 2.0)),
... ])
>>> matches, details = result1.compare_to(result2, relative_threshold=1e-5)
>>> matches
False
>>> "ε^-2" in details
True
```

Parameters
----

other : VakintNumericalResult
   The other numerical result to compare to.
relative_threshold : float
   The relative threshold for comparison.
error : Optional[VakintNumericalResult]
   An optional numerical result representing the error in the evaluation.
max_pull : Optional[float]
   The maximum pull for comparison. Default is 3.0.

other

Kind: Parameter

VakintNumericalResult

The other numerical result to compare to.

relative_threshold

Kind: Parameter

float

The relative threshold for comparison.

error

Kind: Parameter

Optional[VakintNumericalResult]

Default: None

An optional numerical result representing the error in the evaluation.

`max_pull`

Kind: `Parameter`

`Optional[float]`

Default: `None`

`The maximum pull for comparison. Default is 3.0.`

Exhaustive reference · Source

Topologies and dependencies

This version supports the topology patterns and external-tool versions below.

External dependencies

Dependency	Minimum version
`FORM`	`4.2.1`
`pySecDec`	`1.6.4`

Supported topology patterns

Name	Loops	Propagator slots
`I1L`	1	1
`I2L`	2	3
`I2L_pinch_3`	2	3
`I3L`	3	6
`I3L_pinch_6`	3	6
`I3L_pinch_1_6`	3	6
`I3L_pinch_3_6`	3	6
`I3L_pinch_1_3_6`	3	6
`I4L_H`	4	9
`I4L_X`	4	9
`I4L_BMW`	4	8
`I4L_FG`	4	8
`I4L_FG_pinch_2`	4	8
`I4L_FG_pinch_3`	4	8
`I4L_FG_pinch_4`	4	8
`I4L_FG_pinch_7`	4	8
`I4L_FG_pinch_8`	4	8
`I4L_FG_pinch_1_8`	4	8
`I4L_FG_pinch_2_4`	4	8
`I4L_FG_pinch_4_7`	4	8
`I4L_FG_pinch_4_8`	4	8
`I4L_FG_pinch_7_8`	4	8
`I4L_FG_pinch_2_4_7`	4	8
`I4L_FG_pinch_3_4_8`	4	8
`I4L_FG_pinch_4_7_8`	4	8
`I4L_FG_pinch_6_7_8`	4	8
`I4L_FG_pinch_4_6_7_8`	4	8

Version information

Save these identifiers with published results and include them in issue reports so that collaborators can reproduce the same software environment.

- **Documentation channel:** `latest`
- **Documentation route:** `/products/vakint/latest/`
- **Source revision:** `e51747446aa724c3ffac5c42c4103d090c09c6b0`

Package versions and enabled features

- `gammaloop/gammalooprs` — `0.3.4` (`no optional features`)
- `gammaloop/gammaloop-api` — `0.1.0` (`features: cli`)
- `gammaloop/gammaloop` — `0.3.4` (`features: python_api, python_abi, python_stubgen`)
- `linnet/linnet` — `0.17.0` (`features: nodestore-vec, serde, bincode, drawing, symbolica`)
- `linnet/linnet-py` — `0.1.0` (`features: extension-module, abi3-py310`)
- `spenso/spenso` — `0.6.0` (`features: shadowing`)
- `spenso/spenso-macros` — `0.3.2` (`features: shadowing`)
- `spenso/spenso-hep-lib` — `0.1.7` (`no optional features`)
- `spenso/spynso3` — `0.1.2` (`features: python_stubgen`)
- `idenso/idenso` — `0.3.0` (`features: bincode, reference-cases`)
- `idenso/idenso` — `0.3.0` (`features: python, python_stubgen`)
- `vakint/vakint` — `0.1.2` (`no optional features`)
- `vakint/vakint` — `0.1.2` (`features: symbolica_community_module, python_stubgen`)