

Introduction to VIPR CLI

14 November 2025



Sascha Creutzburg

SAXONY.ai

Helm & Walter IT Solutions

Introduction to VIPR CLI

4 November 2025



1. Introduction & Getting Started

The VIPR framework (Versatile Inverse Problem Software Framework)

Two ways for usage:

- CLI: VIPR Core + plugins
- Web-App: GUI + API (uses VIPR Core + plugins)

Codebase

vipr-core (CLI): <https://codebase.helmholtz.cloud/vipr/vipr-core>

vipr-reflectometry-plugin: <https://codebase.helmholtz.cloud/vipr/vipr-reflectorch-plugin>

vipr-api: <https://codebase.helmholtz.cloud/vipr/vipr-api>

vipr-frontend: <https://codebase.helmholtz.cloud/vipr/vipr-frontend>

vipr-framework (web-app): <https://codebase.helmholtz.cloud/vipr/vipr-framework>

VIPR Core and Plugin Installation

- foundation of VIPR
- CLI program, based on Cement CLI framework (<https://builtoncement.com/>)
- minimum requirement Python 3.10

Installation with Reflectometry Plugin

```
python3.10 -m venv myenv
```

```
source myenv/bin/activate
```

```
pip install git+https://codebase.helmholtz.cloud/vipr/vipr-core.git
```

```
pip install git+https://codebase.helmholtz.cloud/vipr/vipr-reflectorch-plugin.git
```

VIPR Core Test with Reflectometry Plugin

Prediction for XRR (PTCDI-C3 layer on SiO_x/Si)

`viPr --config @viPr_reflectometry/reflectorch/examples/configs/PTCDI-C3.yaml inference run`

Terminal Output:

```
INFO: LoadModelInferenceStep completed
INFO: Step 3: Normalizing data
INFO: Starting NormalizeInferenceStep
INFO: No normalizer handler specified, using identity normalization
INFO: NormalizeInferenceStep completed
INFO: Step 4: Preprocessing data
INFO: Starting PreprocessInferenceStep
INFO: Interpolating 1 spectra with Reflectorch
INFO: No preprocessor handler specified, using identity preprocessing
INFO: PreprocessInferenceStep completed
INFO: Step 5: Performing prediction
INFO: Starting PredictionInferenceStep
INFO: Prediction config: {'handler': 'reflectorch_predictor', 'parameters': {'calc_polished_sld_profile': True, 'prior_bounds': {'roughnesses': [[0, 20], [0, 15], [0, 15]], 'slds': [[10, 13], [20, 15], [10, 15]]}, 'se_q_shift': False}}
INFO: Looking for predictor handler: 'reflectorch_predictor'
INFO: Using predictor handler 'reflectorch_predictor'
INFO: Processing batch of 1 spectra with DataSet interface
INFO: Using scalar q resolution from config: None
INFO: Converted flat dict prior_bounds to array with 8 bounds
INFO: PredictionInferenceStep completed
INFO: Step 6: Postprocessing results
INFO: Starting PostprocessInferenceStep
INFO: Processing standard reflectorch UI data for batch_size=1
INFO: Stored standard reflectorch UI data with 1 spectra
INFO: No postprocessor handler specified, returning data as is
INFO: PostprocessInferenceStep completed
INFO: Inference workflow completed successfully
INFO: Collecting 87 buffered log entries
INFO: UI Plugin stored its own data with ID: 38ac6778-6e5c-4140-8db0-841ff245152b
```

Results Directory:

```
> myenv
├── storage
│   ├── huggingface_cache
│   ├── reflectorch
│   └── results
│       └── 38ac6778-6e5c-4140-8db0-841ff245152b
│           ├── diagrams
│           │   ├── reflectivity_primary_Experimental.csv
│           │   ├── reflectivity_primary_Polished.csv
│           │   ├── reflectivity_primary_Predicted.csv
│           │   ├── residuals_vs_q_Polished.csv
│           │   ├── residuals_vs_q_Predicted.csv
│           │   ├── residuals_vs_q_Zero.csv
│           │   ├── sld_profile_Polished_SLD.csv
│           │   └── sld_profile_Predicted_SLD.csv
│           ├── images
│           │   └── reflectivity_matplotlib.png
│           ├── tables
│           │   └── model_parameters.csv
│           ├── data.pkl
│           ├── ! PTCDI-C3.yaml
│           └── metadata.json
```

VIPR CLI command for prediction

`vipr --config @vipr_reflectometry/reflectorch/examples/configs/PTCDI-C3.yaml inference run`

Diagram illustrating the components of the VIPR CLI command:

- `--config`: Launch VIPR
- `@vipr_reflectometry/reflectorch/examples/configs/PTCDI-C3.yaml`: Path to VIPR config (package notation)
- `inference`: Controller Class
- `run`: Method

Config File Reference Options

Package Notation: `--config @vipr_reflectometry/reflectorch/examples/configs/PTCDI-C3.yaml`

Relative Path: `--config ./configs/my-config.yaml`

Absolute Path: `--config /home/user/project/configs/config.yaml`

Introduction to VIPR CLI

4 November 2025



2. VIPR architecture

VIPR Workflow Concept

A workflow processes data through a pipeline (e.g. performing a prediction).

VIPR Core provides:

- Plugin infrastructure
- Handler/Filter/Hook system
- Config-driven component selection

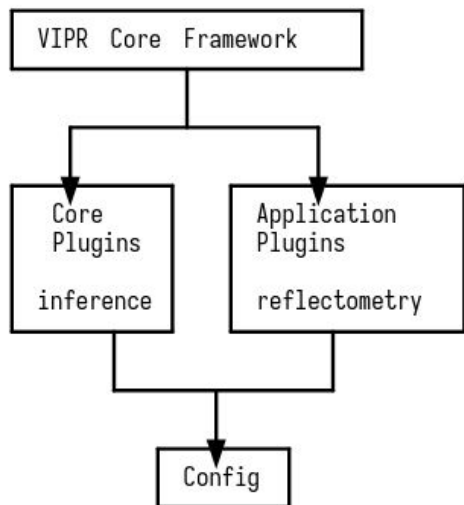
Plugins provide:

- Workflow definitions
- Component implementations
- Built-in and external plugins
- Domain-specific logic in external application plugins

VIPR Config (YAML) connects everything:

- Selects which components to use
- No code changes needed for adjusting the workflows

VIPR Architecture Overview



Plugin Architecture

VIPR Core

- └ Basic framework infrastructure

Core Plugins (Built-in)

- └ `vipr.plugins.inference`
 - └ Defines general inference workflow

Application Plugins (External)

- └ `vipr-reflectometry-plugin`
 - └ Uses Core inference workflow or Overrides
 - └ Registers: Filters for Interpolation
 - └ Implements model loader for reflectometry

Architecture Principles: Component types

1. Interface Implementations (uses Cement handler architecture)

Purpose: Concrete implementations of abstract interfaces

Examples: csv_spectrareader for DataLoader interface, reflectorch for ModelLoader

2. Filters (Transformative callbacks)

Purpose: Transform data (type-preserving output), chainable

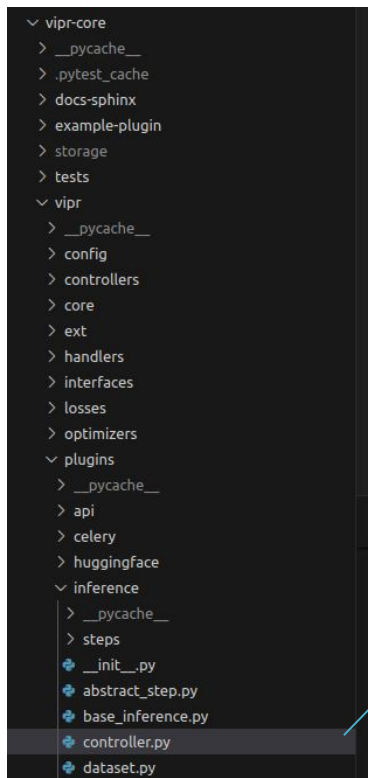
Examples: Interpolation, Normalization

3. Hooks (Read-Only Callbacks)

Purpose: Observe data, no return value, chainable

Example: Logging

InferenceController - The Entry Point for the Prediction



```
controller.py X
1 > from cement import Controller, ex...
3
4 > if TYPE_CHECKING:
5     from vibr.main import VIPR
6
7
8 > class InferenceController(Controller):
9     """Controller for inference operations"""
10
11 > if TYPE_CHECKING:
12     app: VIPR
13
14 > class Meta:
15     label = 'inference'
16     stacked_on = 'base'
17     stacked_type = 'nested'
18
19 @ex(help='Run inference workflow')
20 > def run(self):
21     """
22     Runs the inference workflow.
23     Data comes from self.app.config (loaded from YAML).
24
25     Returns:
26     Dict with inference results
27     """
28     self.app.log.info("Running inference workflow")
29     result = self.app.inference.run()
30     return result
```

Inference Workflow: run() method

1. LoadDataStep

Interface: DataLoader

Handler: csv_spectrareader ← from config

→ Reads Q, I values

2. LoadModelStep

Interface: ModelLoader

Handler: reflectorch ← from config

→ Loads neural network

3. NormalizeStep (optional)

4. PreprocessStep

Filters: interpolation ← from config

→ Prepares data for model

5. PredictionStep

Interface: Predictor

Handler: reflectorch_predictor ← from config

→ Runs inference

6. PostprocessStep

→ Saves results, creates plots ← from config

Simplified Inference process

class Inference:

def run(self):

 # Load

 dataset = LoadDataStep.run()

 model = LoadModelStep.run()

 # Preprocess/Normalize

 dataset = NormalizeStep.run(dataset)

 dataset = PreprocessStep.run(dataset)

 # Predict

 result = PredictionStep.run(dataset, model)

 # Postprocess

 PostprocessStep.run(result)

Hook/Filter Execution Order

Within each Pipeline Step:

1. PRE_PRE_FILTER_HOOK
↓
2. PRE_FILTER (transforms params/data)
↓
3. POST_PRE_FILTER_HOOK
↓
4. [Execute Step - Main Operation]
↓
5. PRE_POST_FILTER_HOOK
↓
6. POST_FILTER (transforms data/result)
↓
7. POST_POST_FILTER_HOOK

Example for LOAD_DATA step

INFERENCE_LOAD_DATA_PRE_PRE_FILTER_HOOK

INFERENCE_LOAD_DATA_PRE_FILTER

INFERENCE_LOAD_DATA_POST_PRE_FILTER_HOOK

[Load data from source] → call DataLoader Handler

INFERENCE_LOAD_DATA_PRE_POST_FILTER_HOOK

INFERENCE_LOAD_DATA_POST_FILTER

INFERENCE_LOAD_DATA_POST_POST_FILTER_HOOK

DataSet - Type-Safe Data Container

Why not just (x, y)?

- ✗ No type safety
- ✗ Mutable (risky in pipelines)

✓ DataSet provides:

- Type safety (Pydantic validation)
- Immutable (read-only arrays)
- Unified structure

Structure:

- x: np.ndarray (batch, n_points)
- y: np.ndarray (batch, n_points)
- dx: Optional[np.ndarray] (uncertainties)
- dy: Optional[np.ndarray] (uncertainties)
- metadata: Dict (file path, etc.)

```
# Creating a DataSet
```

```
from vipr.plugins.inference.dataset import  
DataSet
```

```
# Single spectrum (automatically batched)
```

```
data = DataSet(  
    x=np.array([0.01, 0.02, ..., 0.1]), # (87,)   
    y=np.array([1e-5, 2e-5, ..., 1e-6]), # (87,)   
    metadata={'file': 'PTCDI-C3.txt'}  
)
```

VIPR config

```
vipr:
  inference:
    hooks: ...           ← read-only callbacks (logging,...)
      └─ HOOK_NAME:      ← hook id
          └─ class (fully qualified name)
          └─ method name
          └─ parameters
          └─ enabled: true
    filters: ...         ← transformative callbacks
      └─ FILTER_NAME:    ← filter id
          └─ class (fully qualified name)
          └─ method name
    ...
  load_data:             ← data loader
    └─ HANDLER_NAME:
    └─ parameters: ...
  load_model: ...        ← model loader
  normalize:
  prediction:
  postprocess:
```

```
PTCDI-C3.yaml M
1 vipr:
2   inference:
3     hooks:
4     filters:
5       INFERENCE_PREPROCESS_PRE_FILTER:
6       - class: vipr_reflectometry.reflectorch.reflectorch_extension.Reflectorch
7         enabled: true
8         method: _preprocess_interpolate
9         parameters: null
10        weight: 0
11    load_data:
12      handler: csv_spectrareader
13      parameters:
14        column_mapping:
15          I: 1
16          q: 0
17        data_path: '@vipr_reflectometry/reflectorch/examples/data/PTCDI-C3.txt'
18    load_model:
19      handler: reflectorch
20      parameters:
21        config_name: b_mc_point_xray_conv_standard_L2_InputQ
22  > normalize:-
23  > postprocess:-
24  > prediction:
25    handler: reflectorch_predictor
26    parameters:
27      calc_polished_sld_profile: true
28      calc_pred_curve: true
29      calc_pred_sld_profile: true
30      clip_prediction: true
31      polish_prediction: true
32      prior_bounds:
33        roughnesses: [[0, 20], [0, 15], [0, 15]]
34        slds: [[10, 13], [20, 21], [20, 21]]
35        thicknesses: [[1, 400], [1, 10]]
36      q_resolution: null
37      upper_phase_sld: 0
38      use_q_shift: false
39  > preprocess:-
40  > result_id: 38ac6778-6e5c-4140-8db0-841ff245152b
```

The controller loads this config and executes the workflow

Introduction to VIPR CLI

4 November 2025



3. Integration Example - Reflectorch

Prediction with Reflectorch Package

Load Model

```
inference_model = EasyInferenceModel(config_name='b_mc_point_xray_conv_standard_L2_InputQ',  
                                     model_name=None,  
                                     root_dir=None,  
                                     repo_id='valentinsingularity/reflectivity',  
                                     device='cpu',  
                                     )
```

Load Data

```
data = np.loadtxt('../exp_data/data_PTCDI-C3.txt', delimiter='\t', skiprows=1)  
q_exp = data[:, 0]  
curve_exp = data[:, 1]
```

Preprocess Data

```
q_model, exp_curve_interp = inference_model.interpolate_data_to_model_q(q_exp, curve_exp)  
  
print(q_model.shape, q_model.min(), q_model.max())  
print(exp_curve_interp.shape)
```

Prediction with Reflectorch Package

Predict

```
prior_bounds = [(1., 400.), (1., 10.), #layer thicknesses (top to bottom)
                 (0., 20.), (0., 15.), (0., 15.), #interlayer roughnesses (top to bottom)
                 (10., 13.), (20., 21.), (20., 21.)] #real layer slds (top to bottom)
```

```
prediction_dict = inference_model.predict(
    reflectivity_curve=exp_curve_interp,
    prior_bounds=prior_bounds,
    q_values=q_model,
    clip_prediction=False,
    polish_prediction=True,
    use_q_shift=False,
    calc_pred_curve=True,
)
print(prediction_dict.keys())

pred_params = prediction_dict['predicted_params_array']
pred_curve = prediction_dict['predicted_curve']
```

Reflectometry Plugin - Reflectorch Integration

Shared:

CSV Spectra Reader: [vibr_reflectometry/shared/data_loader/csv_spectrareader_data_loader.py](#)
(Reads XRR data → VIPR DataSet)

Reflectorch-specific:

Model Loader: [vibr_reflectometry/reflectorch/model_loader/reflectorch_model_loader.py](#)
(Loads Reflectorch's EasyInferenceModel)

Interpolation Filter [vibr_reflectometry/reflectorch/reflectorch_extension.py::_preprocess_interpolate](#)
(Adapts the Q-grid of the selected model)

Predictor: [vibr_reflectometry/reflectorch/predictors/reflectorch_predictor.py](#)
(Runs inference)

Postprocessing methods in `data_collectors` directory (plot results)

Summary

Plugin Architecture

- Core framework + application-specific plugins
- Extensible through handlers, filters, and hooks
- Config-driven component selection

Inference Pipeline

6 steps: Load Data → Load Model → Normalize → Preprocess → Predict → Postprocess
Type-safe data handling with DataSet

Example for prediction

```
vipr --config @vipr_reflectometry/reflectorch/examples/configs/PTCDI-C3.yaml inference run
```

Introduction to VIPR CLI

4 November 2025



Appendix

Discovery Core Plugin

Why a discovery registry?

- ✓ Discovery - List available components
- ✓ Web UI - Show dropdown of options
- ✓ Config checking - Validate before execution

Usage:

```
@discover_data_loader('csv_spectrareader')  
@discover_model_loader('reflectorch')  
@discover_filter('INFERENCE_PREPROCESS_PRE_FILTER')  
@discover_predictor('reflectorch_predictor')
```

Examples:

```
viPr discovery components (query for all available components in registry)  
viPr discovery hooks (query for available hooks)  
viPr discovery filters (query for available filters)  
viPr discovery data-loaders  
viPr discovery model-loaders
```

DataCollector - Web-UI & API Integration

Purpose: Collect & structure data for frontend visualization

Builder Pattern for UI Components:

Tables

```
table = dc.table("model_parameters")  
table.add_row(parameter="thickness", predicted="42.5 Å", polished="43.1 Å")
```

Images

```
dc.image("sld_plot").set_from_matplotlib(fig, format="png", dpi=150)
```

Result Storage:

- UUID-based result management
- Saves tables as CSV, images as PNG/SVG
- Accessible via Web-UI and API

VIPR WebApp

README:

<https://codebase.helmholtz.cloud/vipr/vipr-framework/-/blob/init/README.md>

Requirements:

- Access to registry.hzdr.de via Personal Access Token
- Docker

Start WebApp:

```
docker compose up
```