# phaseGP API Reference

### Navigation

## models.py

Gaussian Process models for binary phase classification with active learning and transfer learning capabilities.

### Classes

#### PhaseGP

Variational Gaussian Process model for binary phase classification.

```
PhaseGP(train_x, min_scale=None, max_scale=None, kernel_choice='matern32',
        lengthscale=0.3, learning_rate=0.1, training_iterations=120,
        lengthscale_interval=(0.2,0.3), outputscale_interval=(1.0,4.0))
```

| Parameter | Type | Default | Description |
|---|---|---|---|
| **train_x** | torch.Tensor | required | Initial training inputs used as inducing points, shape (n, d) |
| **min_scale** | array-like | None | Minimum values for input scaling |
| **max_scale** | array-like | None | Maximum values for input scaling |
| **kernel_choice** | str | 'matern32' | Type of kernel: 'rbf', 'matern32', or 'matern52' |
| **lengthscale** | float | 0.3 | Initial kernel lengthscale |
| **learning_rate** | float | 0.1 | Optimization learning rate |
| **training_iterations** | int | 120 | Number of training iterations |
| **lengthscale_interval** | tuple | (0.2, 0.3) | Prior bounds for lengthscale parameter |
| **outputscale_interval** | tuple | (1.0, 4.0) | Prior bounds for outputscale parameter |
| **device** | string | "cpu" or "cuda" | Model's device |

**Methods:**

- `fit(train_x, train_y, epsilon=0.05, verbose=False)` - Train the model using variational inference
- `predict(x)` - Predict binary phase labels
- `predict_proba(x)` - Predict phase probabilities
- `acquisition(x, sampled_points=None, epsilon=0.05, vanilla_acq=False, distance_acq=True, requires_grad=False)` - Compute acquisition values for active learning
- `forward(x)` - Forward pass through the GP model

### PhaseTransferGP

Transfer Learning GP model for phase classification with multiple source models.

```
PhaseTransferGP(source_model_list, train_x, min_scale=None, max_scale=None,
            prior_aggregation="linear", max_adaptive_power=5,
            explorative_threshold=0.4, kernel_choice='matern32',
            lengthscale=0.3, learning_rate=0.1, training_iterations=120,
            lengthscale_interval=(0.2,0.3), outputscale_interval=(1.0,4.0))
```

Additional parameters beyond PhaseGP:

| Parameter | Type | Default | Description |
|---|---|---|---|
| **source_model_list** | list | required | List of pre-trained source models |
| **prior_aggregation** | str | 'linear' | Method for aggregating priors: 'linear' or 'highest' |
| **max_adaptive_power** | float | 5 | Maximum power for adaptive weighting |
| **explorative_threshold** | float | 0.4 | Threshold for exploration vs exploitation (0 to 1) |

**Methods:**

- `fit(train_x, train_y, epsilon=0.05, verbose=False)` - Train the transfer learning model
- `predict(x)` - Predict binary phase labels
- `predict_proba(x)` - Predict phase probabilities using weighted combination
- `get_weight(x)` - Get the weight of the best source model at each input point
- `acquisition(x, sampled_points, epsilon=0.05, vanilla_acq=False, distance_acq=True, requires_grad=False)` - Compute acquisition function for transfer learning
- `forward(x)` - Forward pass combining source and target model predictions

### SKPhaseTransferGP

Transfer Learning GP with scikit-learn compatible source models. Requires scikit-learn source models to have a predict_proba() method.

Inherits from PhaseTransferGP with same parameters. Differs in handling scikit-learn model predictions.

# source_pruner.py

Source model selection algorithms for identifying diverse source models in transfer learning.

## Functions

### source_model_pruner

Select diverse source models from a collection based on phase diagram similarity.

```
source_model_pruner(source_model_list, x_min=None, x_max=None,
                    y_min=None, y_max=None, grid_size=100,
                    acc_threshold=0.9, min_intersection_regions=3,
                    intersection_tol=5, return_index=False)
```

| Parameter | Type | Default | Description |
|---|---|---|---|
| **source_model_list** | list | required | List of trained phase classification models |
| **x_min** | float | None | Minimum x-coordinate for evaluation grid |
| **x_max** | float | None | Maximum x-coordinate for evaluation grid |
| **y_min** | float | None | Minimum y-coordinate for evaluation grid |
| **y_max** | float | None | Maximum y-coordinate for evaluation grid |
| **grid_size** | int | 100 | Number of grid points in each dimension |
| **acc_threshold** | float | 0.9 | Similarity threshold for pixel-wise accuracy |
| **min_intersection_regions** | int | 3 | Maximum disagreement regions for similarity |
| **intersection_tol** | int | 5 | Boundary tolerance in pixels |
| **return_index** | bool | False | If True, also return indices of selected models |

**Returns:** List of selected diverse models, or tuple (selected_models, selected_indices) if return_index=True

### similarity_checker

Check if two phase diagrams are similar based on multiple criteria.

```
similarity_checker(phase_diagram1, phase_diagram2, acc_threshold=0.8,
                   min_intersection_regions=3, intersection_tol=5)
```

| Parameter | Type | Default | Description |
|---|---|---|---|
| **phase_diagram1** | np.ndarray | required | First binary phase diagram |
| **phase_diagram2** | np.ndarray | required | Second binary phase diagram |
| **acc_threshold** | float | 0.8 | Minimum required pixel-wise accuracy (0-1) |
| **min_intersection_regions** | int | 3 | Maximum allowed disagreement regions |
| **intersection_tol** | int | 5 | Dilation radius for boundary tolerance (pixels) |

**Returns:** bool - True if diagrams are similar, False otherwise

### source_diagram_pruner

Select diverse phase diagrams from a collection based on similarity.

```
source_diagram_pruner(phase_diagram_list, acc_threshold=0.9,
                      min_intersection_regions=3, intersection_tol=5)
```

| Parameter | Type | Default | Description |
|---|---|---|---|
| **phase_diagram_list** | list | required | List of 2D binary phase diagram arrays |
| **acc_threshold** | float | 0.9 | Similarity threshold for pixel-wise accuracy |
| **min_intersection_regions** | int | 3 | Maximum disagreement regions for similarity |
| **intersection_tol** | int | 5 | Boundary tolerance in pixels |

**Returns:** list - Selected diverse phase diagrams

# utils.py

Utility functions for data processing, active learning, and general operations.

## Functions

### ensure_tensor

Convert input to PyTorch tensor if not already.

```
ensure_tensor(x, dtype=torch.float32)
```

| Parameter | Type | Default | Description |
|---|---|---|---|
| **x** | array-like | required | Input data (array-like, tensor, or scalar) |
| **dtype** | torch.dtype | torch.float32 | Desired tensor data type |
| **device** | string | "cpu" or "cuda" | Desired device |

**Returns:** torch.Tensor - Input as a PyTorch tensor

### ensure_numpy

Convert input to NumPy array if not already.

```
ensure_numpy(x)
```

| Parameter | Type | Description |
|---|---|---|
| **x** | tensor or array-like | Input data |

**Returns:** np.ndarray - Input as a NumPy array

### brute_sample_new_points

Select new sampling points using brute force evaluation of acquisition function with spacing constraints when batch sampling.

```
brute_sample_new_points(model, candidates, sampled_points=None, n_sample=1,
                    frac_distance_thresh=0.1, epsilon=0.05,
                    vanilla_acq=False, distance_acq=True, return_index=False)
```

| Parameter | Type | Default | Description |
|---|---|---|---|
| **model** | GP model | required | GP model with acquisition method |
| **candidates** | torch.Tensor | required | Candidate points for selection, shape (n, d) |
| **sampled_points** | torch.Tensor | None | Previously sampled points, shape (m, d) |
| **n_sample** | int | 1 | Number of points to select |
| **frac_distance_thresh** | float | 0.1 | Minimum distance between points as fraction of domain |
| **epsilon** | float | 0.05 | Small value for numerical stability in acquisition |
| **vanilla_acq** | bool | False | Whether to use vanilla acquisition function |
| **distance_acq** | bool | True | Whether to include distance-based acquisition component |
| **return_index** | bool | False | If True, also return indices of selected points |

**Returns:** torch.Tensor or tuple - Selected points, or (selected_points, selected_indices) if return_index=True

### gradient_sample_new_points

Select new sampling points using gradient-based optimization of acquisition function with spacing constraints when batch sampling.

```
gradient_sample_new_points(model, sampled_points=None, n_sample=1,
                         frac_distance_thresh=0.1, epsilon=0.05,
                         vanilla_acq=False, distance_acq=True,
                         num_restarts=10, raw_samples=512)
```

| Parameter | Type | Default | Description |
|---|---|---|---|
| **model** | GP model | required | GP model with acquisition method |
| **sampled_points** | torch.Tensor | None | Previously sampled points, shape (m, d) |
| **n_sample** | int | 1 | Number of points to select |
| **frac_distance_thresh** | float | 0.1 | Minimum distance between points as fraction of domain |
| **epsilon** | float | 0.05 | Small value for numerical stability in acquisition |
| **vanilla_acq** | bool | False | Whether to use vanilla acquisition function |
| **distance_acq** | bool | True | Whether to include distance-based acquisition component |

| | | | |
|---|---|---|---|
| **num_restarts** | int | 10 | Number of random restarts for optimization |
| **raw_samples** | int | 512 | Number of raw samples for initialization |

**Returns:** torch.Tensor - Selected points, shape (n_sample, d)

### get_grid

Generate a regular 2D grid for phase diagram evaluation.

```
get_grid(x_min=0, x_max=1, grid_size=100, return_coordinates=False,
         y_min=None, y_max=None)
```

| Parameter | Type | Default | Description |
|---|---|---|---|
| **x_min** | float or list | 0 | Minimum x-coordinate(s). If list, creates N-D grid. |
| **x_max** | float or list | 1 | Maximum coordinate(s). If list, creates N-D grid. |
| **grid_size** | int | 100 | Number of points along each axis |
| **return_coordinates** | bool | False | If True, also return coordinate vectors |
| **y_min** | float | None | Minimum y-coordinate (only for 2D, defaults to x_min) |
| **y_max** | float | None | Maximum y-coordinate (only for 2D, defaults to x_max) |
| **device** | string | "cpu" or "cuda" | Desired device |

**Returns:** torch.Tensor or tuple: - If return_coordinates=False: Grid points, shape (grid_size^N, N) - If return_coordinates=True: (grid_points, coord_list) where coord_list is a list of coordinate vectors for each dimension

### scaler

Scale data to [0,1] range using min-max normalization.

```
scaler(x, min_scale, max_scale)
```

| Parameter | Type | Description |
|---|---|---|
| **x** | torch.Tensor | Input data to scale |
| **min_scale** | torch.Tensor | Minimum values for each dimension |
| **max_scale** | torch.Tensor | Maximum values for each dimension |

**Returns:** torch.Tensor - Scaled data in [0, 1] range

### set_seeds

Set random seeds for reproducibility across PyTorch and NumPy.

```
set_seeds(seed)
```

| Parameter | Type | Description |
|---|---|---|
| **seed** | int | Random seed value |

# visualization.py

Visualization tools for phase diagrams, probability maps, and acquisition functions.

## Functions

### model_diagram_plot

Generate various 2D plots from a trained GP model.

```
model_diagram_plot(model, plot_type="phase", x_min=None, x_max=None,
                   y_min=None, y_max=None, grid_size=100,
                   sampled_points=None, phase_labels=None, title=None,
                   xlabel="Parameter 1", ylabel="Parameter 2",
                   figsize=(7,6), cmap=None, plot_boundary=False)
```

| Parameter | Type | Default | Description |
|---|---|---|---|
| **model** | GP model | required | Trained GP model with predict/predict_proba/acquisition methods |
| **plot_type** | str | "phase" | Type of plot ('phase', 'probability', 'acquisition') |
| **x_min** | float | None | Minimum x-coordinate for plot domain |
| **x_max** | float | None | Maximum x-coordinate for plot domain |
| **y_min** | float | None | Minimum y-coordinate for plot domain |
| **y_max** | float | None | Maximum y-coordinate for plot domain |
| **grid_size** | int | 100 | Number of grid points in each dimension |
| **sampled_points** | torch.Tensor | None | Already sampled points (for acquisition plot) |
| **phase_labels** | list | None | Names for phase 0 and phase 1 |
| **title** | str | None | Plot title |
| **xlabel** | str | "Parameter 1" | Label for x-axis |
| **ylabel** | str | "Parameter 2" | Label for y-axis |
| **figsize** | tuple | (7, 6) | Figure size as (width, height) in inches |

|  |  |  |  |
|---|---|---|---|
| **cmap** | str | None | Colormap name (defaults based on plot_type) |
| **plot_boundary** | bool | False | Whether to show phase boundaries as contour lines |

**Returns:** tuple (fig, ax) - Matplotlib figure and axes objects

### phase_diagram_plot

Plot a phase diagram with optional contour lines at phase boundaries.

```
phase_diagram_plot(phase_diagram, x_coords=None, y_coords=None,
          phase_labels=None, title=None, xlabel="Parameter 1",
          ylabel="Parameter 2", figsize=(7,6), cmap='coolwarm',
          plot_boundary=False)
```

| Parameter | Type | Default | Description |
|---|---|---|---|
| **phase_diagram** | np.ndarray | required | 2D array of phase indices, shape (n_y, n_x) |
| **x_coords** | array-like | None | X-axis coordinate values |
| **y_coords** | array-like | None | Y-axis coordinate values |
| **phase_labels** | list | None | Names for each phase |
| **title** | str | None | Plot title |
| **xlabel** | str | "Parameter 1" | X-axis label |
| **ylabel** | str | "Parameter 2" | Y-axis label |
| **figsize** | tuple | (7, 6) | Figure size (width, height) in inches |
| **cmap** | str/colormap | 'coolwarm' | Colormap for phases |
| **plot_boundary** | bool | False | Whether to draw black contour lines at boundaries |

**Returns:** tuple (fig, ax) - Matplotlib figure and axes objects

### phase_diagram_probability_plot

Plot phase probabilities with decision boundary at p=0.5.

```
phase_diagram_probability_plot(phase_probabilities, x_coords=None,
                  y_coords=None, title=None,
                  xlabel="Parameter 1", ylabel="Parameter 2",
                  figsize=(7,6), cmap='viridis',
                  plot_boundary=True)
```

| Parameter | Type | Default | Description |
|---|---|---|---|

| | | | |
|---|---|---|---|
| **phase_probabilities** | np.ndarray | required | 2D array of probabilities [0,1], shape (n_y, n_x) |
| **x_coords** | array-like | None | X-axis coordinate values |
| **y_coords** | array-like | None | Y-axis coordinate values |
| **title** | str | None | Plot title |
| **xlabel** | str | "Parameter 1" | X-axis label |
| **ylabel** | str | "Parameter 2" | Y-axis label |
| **figsize** | tuple | (7, 6) | Figure size (width, height) in inches |
| **cmap** | str/colormap | 'viridis' | Colormap for probability values |
| **plot_boundary** | bool | True | Whether to highlight the p=0.5 decision boundary |

**Returns:** tuple (fig, ax) - Matplotlib figure and axes objects

### phase_acquisition_plot

Plot acquisition function values for active learning visualization.

```
phase_acquisition_plot(acquisition_values, x_coords=None, y_coords=None,
                title=None, xlabel="Parameter 1",
                ylabel="Parameter 2", figsize=(7, 6),
                cmap='plasma', show_maximum=True)
```

| Parameter | Type | Default | Description |
|---|---|---|---|
| **acquisition_values** | np.ndarray | required | 2D array of acquisition values, shape (n_y, n_x) |
| **x_coords** | array-like | None | X-axis coordinate values |
| **y_coords** | array-like | None | Y-axis coordinate values |
| **title** | str | None | Plot title |
| **xlabel** | str | "Parameter 1" | X-axis label |
| **ylabel** | str | "Parameter 2" | Y-axis label |
| **figsize** | tuple | (7, 6) | Figure size (width, height) in inches |
| **cmap** | str/colormap | 'plasma' | Colormap for acquisition values |
| **show_maximum** | bool | True | Whether to mark the maximum acquisition point |

**Returns:** tuple (fig, ax) - Matplotlib figure and axes objects