Ludwig is a declarative machine learning framework that makes it easy to define machine learning pipelines using a simple and flexible data-driven configuration system. Ludwig is suitable for a wide variety of AI tasks, and is hosted by the Linux Foundation AI & Data.

The configuration declares input and output features, with their respective host data types. Users can also specify additional parameters to preprocess, encode, and decode features, load from pre-trained models, and use the power of ML.

### What is Ludwig?

| What is Ludwig? | Ludwig is a declarative machine learning framework that makes it easy to define machine learning pipelines using a simple and flexible data-driven configuration system. Ludwig is suitable for a wide variety of AI tasks, and is hosted by the Linux Foundation AI & Data.
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Easy to start</td>
<td>Expert level control</td>
</tr>
<tr>
<td>Declerative ML Systems</td>
<td>Advanced functionalities</td>
</tr>
</tbody>
</table>
| Opt-in Complexity | End-to-end machine learning pipelines built using whatever is explicitly specified in the configuration, while falling back to smart defaults for any parameters that are not.
| Opt-in Complexity | End-to-end machine learning pipelines built using whatever is explicitly specified in the configuration, while falling back to smart defaults for any parameters that are not.
| Flexibility | Simplicity |
| Ludwig is suitable for a wide variety of AI tasks, and is hosted by the Linux Foundation AI & Data. | Ludwig is suitable for a wide variety of AI tasks, and is hosted by the Linux Foundation AI & Data.

### Declarative Machine Learning with Ludwig:

- **End-to-end machine learning pipelines using simple and flexible data-driven configurations**
  
  - torchscript: ray horovod huggingface

### One Platform For Research Scientists, Data Scientists, and Engineers

- **For Research Scientists**
  - Minimal machine learning boilerplate: Ludwig takes care of the engineering complexity of deep learning out of the box, enabling research scientists to focus on building models at the highest level of abstraction.
  - Native PyTorch modules: Add a simple decorator to your models implemented as PyTorch module to add it to the Ludwig registry and immediately usable in the Ludwig config.
  - Reproducibility Guarantees: Guarantees the same preprocessing, training and evaluation is performed in both cases, to easily and fairly assess the performance.
  - Hyperparameter optimization with Ray Tune: Hyperparameter optimization process can be run locally or on a Ray cluster, and any of the search algorithms RayTune supports can be chosen, including Bayesian optimization, Hyperband, Nevergrad and others.
  - Built-in datasets: Registered models can be subsequently applied across the extensive set of tasks and datasets that Ludwig supports, or on new ones from a suite of data sources (local files, pandas, dask, modin, or S3).

- **For Data Scientists**
  - Low/no-code interface for state-of-the-art models: Ludwig provides robust implementations of common architectures including CNNs, RNNs, Transformers, TableNet, and MLP-Mixer. Ludwig also natively integrates with pre-trained models.
  - Native integration with Huggingface Transformers: Use Huggingface models by simply specifying them by name in the Ludwig config.
  - Reasonably good AutoML baselines: For tabular and text classification – obtain a reasonably good model by providing a dataset, the target column, and a time budget.
  - Experiment tracking and visualization libraries: Wide variety of data visualizations for model metrics and evaluation results, with integrations with Aim, Tensorboard, MLFlow, and Comet ML.

- **For Engineers**
  - Effortless End-to-End Scale to Multi-Node, Multi-GPU with native Ray Integration: The same Ludwig command line and Python API calls that run on your local laptop can scale across a cluster of machines in the cloud with zero code changes. All you need to do is start a ray cluster and submit your existing Ludwig command or script to run the Ray CLI. Ludwig handles the end-to-end orchestration and distributed execution automatically. Dask on Ray will be used to scale preprocessing to arbitrarily large datasets.
  - Reproducibility Guarantees: Guarantees the same preprocessing, training and evaluation is performed in both cases, to easily and fairly assess the performance.
  - Hyperparameter optimization with Ray Tune: Hyperparameter optimization process can be run locally or on a Ray cluster, and any of the search algorithms RayTune supports can be chosen, including Bayesian optimization, Hyperband, Nevergrad and others.
  - Built-in datasets: Registered models can be subsequently applied across the extensive set of tasks and datasets that Ludwig supports, or on new ones from a suite of data sources (local files, pandas, dask, modin, or S3).

### How does it work?

Data-driven config-based ML orchestrations with Python APIs and CLI

```
> pip install ludwig
```

### How to use it?

**Out of the box:** Distributed training

```
> ray start --address localhost:6379
> ray stop
```

```
> LudwigModel.load(ludwig_model_path)
```

**Out of the box:** Huggingface Models

```
> ludwig train --dataset sst5 --config str
```

```
> (input_features: {name: sentence, type: text, encoder: bert})
> output_features: {name: label, type: category})
```

**Out of the box:** Torchscript-powered REST APIs

```
> ludwig export_torchscript -m=results/experiment_run/model
> ludwig serve --model_path=PATH/TO/model
> curl http://0.0.0.0:8000/predict -F "english_text=words to be translated"
```

**Stay Connected and Help Contribute!**

- Ludwig slack
- GitHub
- Facebook
- Twitter
- Join the Ludwig Slack

**Find more examples and full documentation at ludwig.ai**

### Declarative Machine Learning with Ludwig:

- **End-to-end machine learning pipelines using simple and flexible data-driven configurations**
  
  - torchscript: ray horovod huggingface

### One Platform For Research Scientists, Data Scientists, and Engineers

- **For Research Scientists**
  - Minimal machine learning boilerplate: Ludwig takes care of the engineering complexity of deep learning out of the box, enabling research scientists to focus on building models at the highest level of abstraction.
  - Native PyTorch modules: Add a simple decorator to your models implemented as PyTorch module to add it to the Ludwig registry and immediately usable in the Ludwig config.
  - Reproducibility Guarantees: Guarantees the same preprocessing, training and evaluation is performed in both cases, to easily and fairly assess the performance.
  - Hyperparameter optimization with Ray Tune: Hyperparameter optimization process can be run locally or on a Ray cluster, and any of the search algorithms RayTune supports can be chosen, including Bayesian optimization, Hyperband, Nevergrad and others.
  - Built-in datasets: Registered models can be subsequently applied across the extensive set of tasks and datasets that Ludwig supports, or on new ones from a suite of data sources (local files, pandas, dask, modin, or S3).

- **For Data Scientists**
  - Low/no-code interface for state-of-the-art models: Ludwig provides robust implementations of common architectures including CNNs, RNNs, Transformers, TableNet, and MLP-Mixer. Ludwig also natively integrates with pre-trained models.
  - Native integration with Huggingface Transformers: Use Huggingface models by simply specifying them by name in the Ludwig config.
  - Reasonably good AutoML baselines: For tabular and text classification – obtain a reasonably good model by providing a dataset, the target column, and a time budget.
  - Experiment tracking and visualization libraries: Wide variety of data visualizations for model metrics and evaluation results, with integrations with Aim, Tensorboard, MLFlow, and Comet ML.

- **For Engineers**
  - Effortless End-to-End Scale to Multi-Node, Multi-GPU with native Ray Integration: The same Ludwig command line and Python API calls that run on your local laptop can scale across a cluster of machines in the cloud with zero code changes. All you need to do is start a ray cluster and submit your existing Ludwig command or script to run the Ray CLI. Ludwig handles the end-to-end orchestration and distributed execution automatically. Dask on Ray will be used to scale preprocessing to arbitrarily large datasets.
  - Reproducibility Guarantees: Guarantees the same preprocessing, training and evaluation is performed in both cases, to easily and fairly assess the performance.
  - Hyperparameter optimization with Ray Tune: Hyperparameter optimization process can be run locally or on a Ray cluster, and any of the search algorithms RayTune supports can be chosen, including Bayesian optimization, Hyperband, Nevergrad and others.
  - Built-in datasets: Registered models can be subsequently applied across the extensive set of tasks and datasets that Ludwig supports, or on new ones from a suite of data sources (local files, pandas, dask, modin, or S3).

### How does it work?

Data-driven config-based ML orchestrations with Python APIs and CLI

```
> pip install ludwig
```

### How to use it?

**Out of the box:** Distributed training

```
> ray start --address localhost:6379
> ray stop
```

```
> LudwigModel.load(ludwig_model_path)
```

**Out of the box:** Huggingface Models

```
> ludwig train --dataset sst5 --config str
```

```
> (input_features: {name: sentence, type: text, encoder: bert})
> output_features: {name: label, type: category})
```

**Out of the box:** Torchscript-powered REST APIs

```
> ludwig export_torchscript -m=results/experiment_run/model
> ludwig serve --model_path=PATH/TO/model
> curl http://0.0.0.0:8000/predict -F "english_text=words to be translated"
```

**Stay Connected and Help Contribute!**

- Ludwig slack
- GitHub
- Facebook
- Twitter
- Join the Ludwig Slack

**Find more examples and full documentation at ludwig.ai**