

Ray Wrapper

mlumina.ray_wrapper

Библиотека готовых пайплайнов Ray Train/Tune ("cookbook") + набор датасет-хелперов. В отличие от остальных пакетов, здесь не одна обёртка, а набор функций `train_<model>/tune_<model>` под конкретные типы моделей: `lin_reg`, `svm`, `fcnn`, `cnn` (CIFAR-10), `vit` (CIFAR-10 и CIFAR-100).

Структура (`src/mlumina/ray_wrapper/`):

- `data/ray_data_loader.py` — построение датасетов для Ray Train/Tune.
- `tune/tune_search.py` — фабрика поисковых алгоритмов Ray Tune.
- `models/{lin_reg,svm,fcnn,cnn,vit}/...` — сами модели и их `train_*/tune_*` функции.
- `examples/{lin_reg,svm,fcnn,cnn,vit}/...` — запускные скрипты: `train.py/tune.py` (запуск локально против Ray-кластера) и `schedule_*.py` (та же задача, но через [scheduler](#)).

`mlumina.ray_wrapper.__init__` реэкспортирует: `build_search_algorithm`, всё из `.data` (датасет-хелперы), всё из `.models` (`CNN`, `train_cnn`, `tune_cnn`, `FCNN`, `train_fcnn`, `tune_fcnn`, `tune_lin_reg`, `tune_svm`, ViT-функции).

???????? (ray_wrapper.data)

```
from mlumina.ray_wrapper.data import (
    build_torchvision_dataset,      # mnist / cifar10 / cifar100 -> torchvision Dataset
    build_image_ray_datasets,       # -> {"train": ray.data.Dataset, "valid":
ray.data.Dataset}
    build_sklearn_ray_datasets,     # diabetes / breast_cancer (sklearn toy-датасеты) ->
{"train": ..., "valid": ...}
    ray_dataset_to_numpy,           # ray.data.Dataset -> (X, y) как numpy
    get_image_dataset_shards_for_worker,
    iter_image_batches,
    unpack_image_batch,
    add_dataset_sizes_to_train_config,
```

```

get_num_batches_for_current_worker,
enable_ray_data_progress_ui,
)

```

????????? ?????????? ??? Ray Tune

```

from mlumina.ray_wrapper import build_search_algorithm

search_alg = build_search_algorithm(
    search_algorithm="optuna", # "tune" (None/дефолтный поиск Ray Tune) | "optuna" |
    "hyperopt"
    metric="rmse",
    mode="min",
    search_algorithm_config=None,
)

```

`build_search_algorithm(search_algorithm="tune", metric="accuracy", mode="max", search_algorithm_config=None)` — фабрика: `"tune"` → `None` (дефолтный семплер Ray Tune), `"optuna"` → `OptunaSearch(...)`, `"hyperopt"` → `HyperOptSearch(...)`. Неизвестное значение бросает `ValueError`.

???????

Самый простой пример-шаблон — `lin_reg` (`src/mlumina/ray_wrapper/models/lin_reg/lin_reg.py`):

```

from mlumina.ray_wrapper import tune_lin_reg

results = tune_lin_reg(
    search_space={
        "fit_intercept": tune.choice([True, False]),
        "positive": tune.choice([True, False]),
    },
    num_samples=20,
    resources={"CPU": 1},
)

best = results.get_best_result(metric="rmse", mode="min")

```

Внутри `tune_lin_reg` каждый trial самостоятельно строит датасет `diabetes` через `build_sklearn_ray_datasets`, обучает `sklearn.linear_model.LinearRegression`, репортит `rmse` / `r2`

через `tune.report`, отбор — `ASHAScheduler`. `svm` устроен аналогично (`sklearn.svm`).

`fcnn`/`cnn`/`vit` — то же самое, но на PyTorch (полносвязная сеть, CNN и Vision Transformer соответственно, на MNIST/CIFAR-10/CIFAR-100) и с парой функций каждая: `train_<model>(...)` — одиночный/распределённый запуск через `ray.train`, `tune_<model>(...)` — подбор гиперпараметров через `ray.tune.Tuner`. Конкретные параметры смотреть в исходниках — `src/mlumina/ray_wrapper/models/{fcnn,cnn/cnn_cifar_10,vit/cifar_10/vit_cifar_10,vit/cifar_100/vit_cifar_100}.py` — и в зеркальных запускных скриптах `src/mlumina/ray_wrapper/examples/<model>/{train,tune}.py`.

?????? scheduler ??????

??????? Ray

Каждая модель в `ray_wrapper/examples/` имеет пару: `train.py`/`tune.py` подключаются к кластеру напрямую (`ray.init(address="auto")`) и выполняются в текущем процессе, а `schedule_*.py` вместо этого отправляет ту же задачу в очередь [scheduler](#)-сервиса:

```
import requests

SCHEDULER_URL = "http://127.0.0.1:8264/add_task"

task_json = {
    "name": "fcnn",
    "entrypoint": "python ray_wrapper/examples/fcnn/tune.py",
    "runtime_env": {
        "working_dir": "/home/ray/src/mlumina",
        "pip": ["ray", "torch", "tqdm"],
    },
    "priority": 0,
    "cpu": 0,
    "gpu": 0,
    "memory": 1,
    "timeout": 0,
}

response = requests.post(SCHEDULER_URL, json=task_json, timeout=60)
```

`working_dir` в `runtime_env` указывает на путь внутри контейнера `scheduler` (`/home/ray/src/mlumina`, см. `docker-compose`-сервис `scheduler`), поэтому такие скрипты рассчитаны на запуск изнутри Docker-стека, а не с произвольной локальной машины.

Revision #1

Created 2026-09-23 06:40:10 UTC by Маслов Алексей

Updated 2026-09-23 06:40:11 UTC by Маслов Алексей