Getting Started

?????? ??????

?????????

Из приватного GitLab (требуется personal access token с правом `read_repository`):

```
pip install "git+https://<username>:<token>@gitlab.mephi-ai.ru/framework/mlumina.git"
```

Для разработки — клонировать репозиторий и установить в editable-режиме:

```
pip install -e .
```

или через `uv` (в репозитории есть `uv.lock`):

```
# CPU-сборка torch
uv sync --extra cpu

# GPU-сборка torch (CUDA 12.8)
uv sync --extra cu128
```

Требуется Python >= 3.12. Экстра `cpu` и `cu128` взаимоисключающие (см. `[tool.uv.conflicts]` в `pyproject.toml`).

> В пакете **нет** публичного `mlumina.__init__` API — `src/mlumina/__init__.py` пуст. Импортировать нужно из конкретных подпакетов: `from mlumina.xgboost import ...`, `from mlumina.core import ...` и т.д. `python -m mlumina` тоже ничего не делает — `__main__.py` пустой стаб.

???????

Рабочие примеры лежат в `examples/`, каждый — в своей подпапке по модулю (`xgboost/`, `image_models/`, `interpolation_tables/`, `torch_models/`, `components_wear/`, `scheduler/`). Корневой `run.sh` перечисляет варианты запуска, но большинство строк там закомментированы — актуальный на данный момент рабочий пример это `examples/components_wear/example_objects_simulate_ray.py`. Остальные примеры нужно запускать вручную и проверять — они не покрыты CI.

Самый полный и надёжный пример — `examples/xgboost/example.py`: загружает конфиг через `mlumina.core`, обучает и тюнит модель локально и через Ray.

????????????? (`mlumina.core`)

Конфигурация собирается через `pydantic-settings` из YAML-файла и CLI-аргументов.

```
from mlumina.core import MLuminaConfig, init_config


config = init_config(MLuminaConfig)


print(config.cluster.head_host)
print(config.data.train_file)
```

Путь к YAML задаётся переменной окружения `MLUMINA_SETTINGS_PATH`, по умолчанию `./configs/settings.yaml` (путь читается относительно текущей рабочей директории процесса). Пример файла — `configs/settings.yaml`:

```
training:
  xgboost:
    objective: reg:squarederror
    tree_method: hist
    subsample: 0.8
    colsample_bytree: 0.8
    num_boost_round: 100
    error_func: rmse
  ray_running:
    num_actors: 2
    cpus_per_actor: 1
    gpus_per_actor: 0


cluster:
  head_host: localhost:10001
  mlflow_uri: http://mlflow-server:5001
```

```
data:
  train_file: examples/xgboost/data/test_xgboost.csv
  eval_file: examples/xgboost/data/test_xgboost.csv
```

Структура `MLuminaConfig` (`src/mlumina/core/config.py`):

- `cluster: ClusterSettings` — `head_host`, `mlflow_uri`.
- `data: DataConfig` — `train_file`, `eval_file`. Оба поля типа `pydantic.FilePath` — **файлы должны существовать на диске**, иначе `init_config` упадёт с `ValidationError`.
- `training: TrainingConfig` — `xgboost: XGBoostTrainSettings | None`, `ray_running: RayRunningSettings | None`.
- `tune: TuneConfig` — `xgboost: XGBoostTuneSettings` (диапазоны гиперпараметров для Optuna/Ray Tune, есть значения по умолчанию).

Важные детали:

- `MLuminaConfig` наследует `cli_parse_args=True` — при создании экземпляра pydantic-settings также парсит `sys.argv`. Неизвестные аргументы игнорируются (`extra="ignore"`), но это стоит иметь в виду при запуске скриптов с собственными CLI-флагами.
- **`import mlumina.core` имеет побочные эффекты**: `src/mlumina/core/__init__.py` сразу вызывает `init_config(MLuminaConfig)` и `setup_logging()`. Это значит, что просто импортировать что-то из `mlumina.core` уже требует наличия валидного `configs/settings.yaml` (или `MLUMINA_SETTINGS_PATH`) с существующими файлами данных.

??????????

```
import logging
from mlumina.core import setup_logging, LogFormat

setup_logging(log_format=LogFormat.JSON, level="DEBUG")  # вызывается автоматически при
импорте mlumina.core

logger = logging.getLogger("mlumina")
logger.info("Обучение модели запущено")
```

Через переменные окружения:

```
export MLUMINA_LOG_FORMAT=json   # json | standard
export MLUMINA_LOG_LEVEL=INFO
```

`setup_logging` также настраивает логгеры `ray`, `ray.data`, `ray.tune`, `ray.rllib`, `ray.train`, `ray.serve`, `ray.workflow` под тот же формат и выставляет `RAY_LOG_TO_STDERR=1`. Вызывать её стоит до `ray.init()`, если нужна интеграция логов Ray.

????????? ?????? ????????????? (Ray/MLflow/MinIO/MySQL/scheduler)

См. deployment.md. Коротко:

```
docker compose up -d --build --remove-orphans
```

???????? ????

```
pre-commit install --install-hooks
pre-commit run --all-files
```

Линтер — `ruff` (конфигурация в `pyproject.toml`, `target-version = "py312"`, `line-length = 119`).

Revision #1
Created 2026-09-23 06:40:02 UTC by Маслов Алексей
Updated 2026-09-23 06:40:03 UTC by Маслов Алексей