

MLKit CLI — Zero-Path Execution Specification

The MLKit CLI operates without requiring manual file or directory paths as arguments. By executing commands from within a project root, the CLI reads `mlkit.json` directly and resolves all project files, directories, datasets, checkpoints, and module entry points automatically.

1. Zero-Path Architecture Principle

- **Root Context Discovery:** Every command expects to run in the directory containing `mlkit.json`.
 - **Implicit Module Resolution:** Based on the "entrypoint" declared in `mlkit.json` (e.g., `my_ai`), the CLI automatically imports and executes the respective subclasses without user-provided file paths:
 - Config: `my_ai/config.py`
 - Data: `my_ai/data.py`
 - Model: `my_ai/model.py`
 - Training: `my_ai/trainer.py`
 - Evaluation: `my_ai/evaluator.py`
 - Serving & Run: `my_ai/inference.py`
 - **Artifact Path Convention:** Training logs, checkpoints, and evaluation metrics resolve to convention-based directories (`data/`, `artifacts/`, `checkpoints/`) without command-line parameter overrides.
-

2. Essential Commands (3-Week MVP Scope)

`mlkit init`

- **Purpose:** Scaffolds a new project directory and creates the initial project layout, configuration manifest, and baseline boilerplate code.
 - **Input Parameters:** Prompts interactively for the project name and task type.
 - **Internal Action:** Generates `mlkit.json` alongside the convention directory structure (`config.py`, `data.py`, `model.py`, `trainer.py`, `evaluator.py`, `inference.py`).
-

`mlkit train`

- **Purpose:** Runs the project's model training cycle.
- **CLI Syntax:**

1.

Shell

```
mlkit train
```

- **Execution Flow:**
 - 2. Locates and reads `mlkit.json` to load project metadata and hardware settings.
 - 3. Resolves and calls `data.py` (`Dataset.load()`) using the source defined in configuration.
 - 4. Instantiates `model.py` (`Model`).
 - 5. Invokes `trainer.py` (`BaseTrainer.fit(model, dataset)`).
 - 6. Serializes the resulting model weights and metadata into the project's default artifact store via `BaseTrainer.checkpoint()` without needing an explicit output path.
-

mlkit evaluate

- **Purpose:** Assesses model performance against validation or test data partitions.
- **CLI Syntax:**

1.

Shell

```
mlkit evaluate
```

- **Execution Flow:**
 - 2. Automatically locates the latest saved model checkpoint from the artifacts directory.
 - 3. Calls `model.py` (`Model.load()`) to restore model state.
 - 4. Calls `data.py` (`Dataset.split()`) to extract the evaluation partition.
 - 5. Invokes `evaluator.py` (`BaseEvaluator.evaluate(model, dataset)`).
 - 6. Outputs the computed metrics (accuracy, loss, F1) to the console.
-

mlkit serve

- **Purpose:** Launches a local HTTP inference server.
- **CLI Syntax:**

1.

Shell

```
mlkit serve
```

- **Execution Flow:**
- 2. Discovers and loads the active trained model instance from the project artifacts directory.
- 3. Spawns an internal HTTP server (e.g., standard port `8000`).
- 4. Binds incoming prediction requests to `inference.py` (`BaseInference.run(model, raw_input)`).
- 5. Streams prediction responses back as JSON without requiring custom routing parameters from

the developer.

mlkit data validate

- **Purpose:** Verifies dataset integrity, column types, and schema readiness before training runs.
- **CLI Syntax:**

1.

Shell

```
mlkit data validate
```

- **Execution Flow:**
2. Resolves `data.py` (`Dataset`) from the project configuration.
 3. Executes `Dataset.load()` followed by `Dataset.validate()`.
 4. Prints schema verification reports and partition summaries directly to the terminal.
-

mlkit doctor

- **Purpose:** Inspects the development environment and verifies hardware accelerator status.
- **CLI Syntax:**

1.

Shell

```
mlkit doctor
```

- **Execution Flow:**
2. Parses `mlkit.json` via `config.py` (`BaseConfig`).
 3. Runs `BaseConfig.resolve_device()` to test hardware visibility (CUDA, Apple Silicon MPS, or CPU fallback).
 4. Asserts local folder accessibility and prints the runtime status report.