The Journal of Open Source Software

Lighter: Configuration-Driven Deep Learning

Ibrahim Hadzic [1,2¶], **Suraj Pai** [2,3,4], **Keno Bressem** [5,6], **Borek Foldyna** [1], **and Hugo JWL Aerts** [2,3,4]

1 Cardiovascular Imaging Research Center, Massachusetts General Hospital, Harvard Medical School, United States of America **2** Radiology and Nuclear Medicine, CARIM & GROW, Maastricht University, The Netherlands **3** Artificial Intelligence in Medicine (AIM) Program, Mass General Brigham, Harvard Medical School, Harvard Institutes of Medicine, United States of America **4** Department of Radiation Oncology, Brigham and Women's Hospital, Dana-Farber Cancer Institute, Harvard Medical School, United States of America **5** Technical University of Munich, School of Medicine and Health, Klinikum rechts der Isar, TUM University Hospital, Germany **6** Department of Cardiovascular Radiology and Nuclear Medicine, Technical University of Munich, School of Medicine and Health, German Heart Center, TUM University Hospital, Germany ¶ Corresponding author

DOI: 10.xxxxxx/draft

Software

- Review
- Repository
- Archive

Editor: Open Journals

Reviewers:

- @openjournals

Submitted: 01 January 1970
Published: unpublished

Summary

Lighter is an open-source Python framework for deep learning research that builds upon PyTorch Lightning (Falcon & The PyTorch Lightning team, 2019) and the MONAI Bundle configuration (Cardoso et al., 2022). With its declarative YAML-based configuration system that is both transparent and self-documenting, Lighter aims to streamline deep learning research. Researchers define experimental protocols—including neural network architectures, optimization strategies, data pipelines, and evaluation metrics—through structured configuration files, effectively decoupling scientific hypotheses from implementation details. This separation reduces boilerplate code while preserving complete experimental control. Lighter enables reproducibility through comprehensive configuration snapshots that document all experimental parameters and dependencies. A modular adapter system and support for project-specific extensions ensure the extensibility of the framework enabling researchers to implement specialized methodologies without modifying core framework components. By abstracting the engineering complexities of deep learning experimentation, Lighter allows researchers to focus on scientific innovation, accelerate hypothesis testing, and facilitate rigorous validation of research findings across application domains.

Statement of Need

Lighter is designed to address several key challenges in deep learning experimentation:

1. **Boilerplate Code:** Writing code for training loops, data loading, metric calculations, and experiment setups is repetitive and can vary greatly between projects. *Lighter abstracts these repetitive tasks, exposing only the components that differ across projects.*

2. **Experiment Management:** Handling numerous hyperparameters and configurations across various experiments can become cumbersome and error-prone. *Lighter offers organized configuration through YAML files, providing a **centralized record of all experiment parameters**.*

3. **Reproducibility:** Reproducing experiments from different implementations can be challenging. *Lighter's **self-contained configuration files** serve as comprehensive documentation, facilitating the exact recreation of experimental setups.*

4. **Collaboration:** Collaborating on experiments often requires understanding complex

codebases. *Lighter enhances collaboration by using standardized configurations, making it easier to share and reuse experiment setups within and across research teams.*

5. **Slowed Iteration:** The cumulative effect of these challenges slows down the research iteration cycle. *Lighter accelerates iteration by streamlining the experiment setup process, allowing researchers to focus on core experiment choices without being bogged down by infrastructure concerns.*

Design

Lighter is built upon three fundamental components:

1. **`Config`**: serves as the experiment's blueprint, parsing and validating YAML configuration files that comprehensively define all aspects of the experimental setup. Within these configuration files, researchers specify the `System` and `Trainer` parameters, creating a self-documenting record of the experiment.
2. **`System`**: orchestrates the main building blocks of an experiment: model, optimizer, scheduler, loss function, evaluation metrics, and dataloaders. It implements the logic controlling how these components interact during training, validation, testing, and inference phases, that is modifiable through adapters.
3. **`Trainer`**: PyTorch Lightning's `Trainer` handles technical aspects like distributed GPU training, mixed precision computation, and checkpoint management. Lighter uses it to execute the experimental protocol defined by the System.

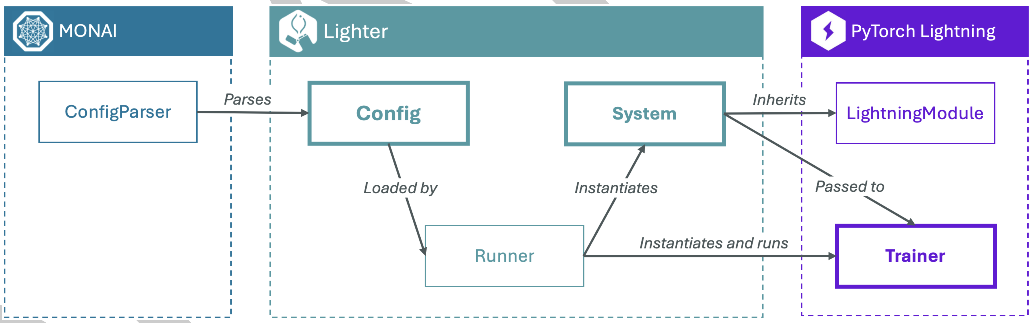

Figure 1: Lighter Overview. Lighter revolves around three main components – `Trainer`, `System` and `Config`, which contains the definition for the former two. `Config` leverages MONAI's `ConfigParser` for parsing the user-defined YAML configuration files, and its features are used by Runner to instantiate the `System` and `Trainer`. `Trainer` is used directly from PyTorch Lightning, whereas `System` inherits from `LightningModule`, ensuring its compatibility with `Trainer` while implementing a logic generalizable to any task or type of data. Finally, `Runner` runs the paired `Trainer` and `System` for a particular stage (e.g., `fit` or `test`).

System

The `System` encapsulates experimental elements—neural network architectures, optimization strategies, learning rate schedulers, loss functions, evaluation metrics, and data loaders. It implements a generalized data flow that orchestrates interactions between these elements during experiments. Through the adapter mechanism, `System` provides flexibility for diverse research tasks without architectural modifications (Figure 2).

The Journal of Open Source Software

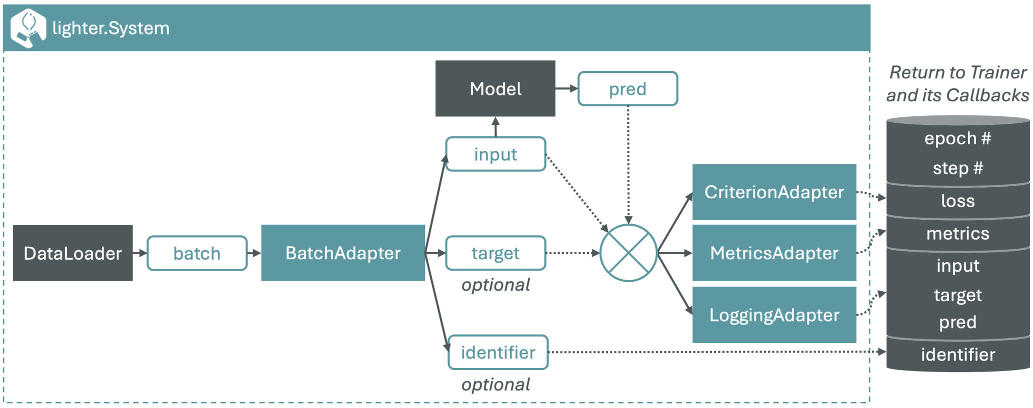

Figure 2: Flowchart of the `lighter.System`. A `batch` from the `DataLoader` is processed by `BatchAdapter` to extract `input`, `target` (optional), and `identifier` (optional). The `Model` generates `pred` (predictions) from the `input`. `CriterionAdapter` and `MetricsAdapter` compute loss and metrics, respectively, by applying optional transformations and adapting arguments for the loss and metric functions. Argument adaptation reorders or names inputs; for example, if a loss function expects `loss_fn(predictions, ground_truth)`, the `CriterionAdapter` maps `pred` to `predictions` and `target` to `ground_truth`. `LoggingAdapter` prepares data for logging. Results, including loss, metrics, and processed data, are returned to the `Trainer`.

Adaptability Through Modular Design

Lighter achieves task-agnostic flexibility through two key concepts: adapters and project-specific module integration.

Adapters

The adapter pattern implements an interface layer between core system components, enabling customized data flow between the dataloader, criterion, metrics computation, and logging subsystems (Figure 2). For example, to apply a sigmoid activation on predictions and to pass the predictions and target data to the appropriate criterion arguments, you can configure the criterion adapter as follows:

```yaml
adapters:
    train:
        criterion:
            _target_: lighter.adapters.CriterionAdapter
            pred_transforms:   # Apply sigmoid activation to predictions
                _target_: torch.sigmoid
            pred_argument: 0   # Pass 'pred' to criterion's 1st arg
            target_argument: 1 # Pass 'target' to criterion's 2nd arg
```

This abstraction layer allows researchers to modify how components interact without changing the core framework code, enabling Lighter to flexibly support diverse tasks, ranging from standard classification to self-supervised learning, through simple adapter configuration changes.

Project-specific modules

Lighter provides the integration of project-specific implementations through a modular project structure. Researchers can use their custom components—including novel architectures, specialized datasets, task-specific metrics, and domain-adapted transforms—within a structured project directory. This organization promotes code reusability and maintains a clear separation between framework functionality and project-specific implementations.

For example, given a project folder `joss_project` with the following structure:

```
joss_project
├── __init__.py
└── models/
    ├── __init__.py
    └── mlp.py
```

This folder will be imported as a module named `project`, which can then be used to reference the components defined within it:

```yaml
project: /path/to/joss_project
system:
    model:
        _target_: project.models.mlp.MLP
        input_size: 784
        num_classes: 10
```

Research Contributions That Use Lighter

Lighter has enabled advancements in medical imaging research:

- Foundation model for cancer imaging biomarkers (Pai et al., 2024)
- Vision Foundation Models for Computed Tomography (Pai et al., 2025)

Acknowledgments

We thank John Zielke for the idea to use adapter design pattern. We thank the MONAI team (Wenqi Li, Nic Ma, Yun Liu, Eric Kerfoot) for their continuous support with features and improvements related to MONAI Bundle.

References

Cardoso, M. J., Li, W., Brown, R., Ma, N., Kerfoot, E., Wang, Y., Murray, B., Myronenko, A., Zhao, C., Yang, D., Nath, V., He, Y., Xu, Z., Hatamizadeh, A., Zhu, W., Liu, Y., Zheng, M., Tang, Y., Yang, I., … Feng, A. (2022). *MONAI: An open-source framework for deep learning in healthcare*. https://doi.org/10.48550/arXiv.2211.02701

Falcon, W., & The PyTorch Lightning team. (2019). *PyTorch Lightning* (Version 1.4). https://doi.org/10.5281/zenodo.3828935

Pai, S., Bontempi, D., Hadzic, I., Prudente, V., Sokač, M., Chaunzwa, T. L., Bernatz, S., Hosny, A., Mak, R. H., Birkbak, N. J., & Aerts, H. J. W. L. (2024). Foundation model for cancer imaging biomarkers. *Nat. Mach. Intell.*, *6*(3), 354–367. https://doi.org/10.1038/s42256-024-00807-9

Pai, S., Hadzic, I., Bontempi, D., Bressem, K., Kann, B. H., Fedorov, A., Mak, R. H., & Aerts, H. J. W. L. (2025). *Vision foundation models for computed tomography*. https://doi.org/10.48550/arXiv.2501.09001