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Published: unpublished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 deep learning framework that builds upon PyTorch Lightning (Falcon & The PyTorch Lightning team, 2019) and MONAI Bundle configuration (Cardoso et al., 2022). It streamlines deep learning research through YAML-based configuration that decouples experiment setup from implementation details. Researchers define models, datasets, and other components via structured configuration files, reducing boilerplate while maintaining control. The framework enhances reproducibility through configuration snapshots and supports extensibility via adapters and project-specific modules. By abstracting engineering complexities, Lighter allows researchers to focus on innovation, accelerate hypothesis testing, and facilitate rigorous validation across 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.*

The Journal of Open Source Software

Design

Lighter is built upon three fundamental components (Figure 1):

1. **`Config`**: serves as the experiment's blueprint, parsing and validating YAML configuration files that 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`**: encapsulates the model, optimizer, scheduler, loss function, metrics, and dataloaders. Importantly, it implements the flow between them that can be customized through adapters (Figure 2).
3. **`Trainer`**: PyTorch Lightning's `Trainer` handles aspects like distributed or mixed-precision training and checkpoint management. Lighter uses it to execute the protocol defined by the `System`.

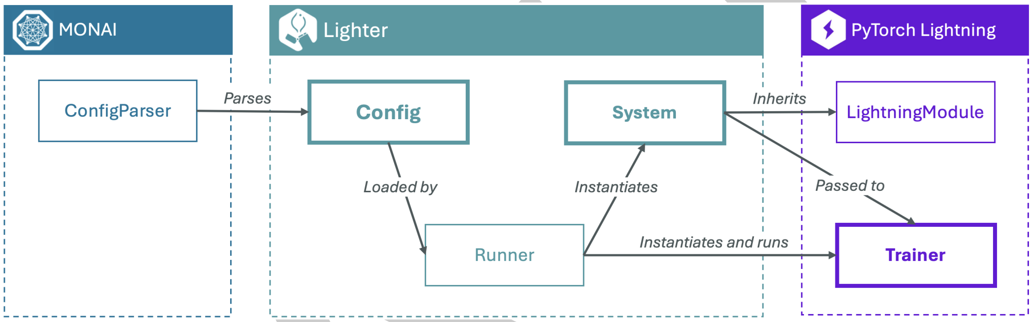

Figure 1: Lighter Overview. `Config` leverages MONAI's `ConfigParser` for parsing the user-defined YAML configuration files, and its features are used by `Runner` to instantiate the `System` and `Trainer`. `Trainer` is used directly from PyTorch Lightning, whereas `System` inherits from `LightningModule`, 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).

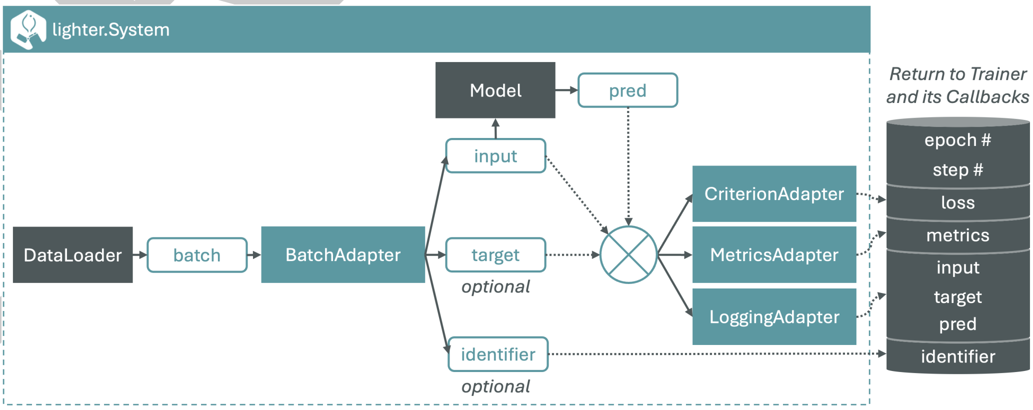

Figure 2: Flowchart of the `lighter.System`. A `batch` from the `DataLoader` is processed by `BatchAdapter` to extract `input`, `target` (optional), and `identifier` (optional). The `Model` generates `pred` (predictions) from the `input`. `CriterionAdapter` and `MetricsAdapter` compute loss and metrics, respectively, by applying optional transformations and routing arguments for the loss and metric functions. Results, including loss, metrics, and other data prepared for logging by the `LoggingAdapter` are returned to the `Trainer`.

Adaptability Through Modular Design

Adapters

The adapter pattern creates an interface between core system components, allowing customization of the data flow. By configuring adapters, users can modify how components interact without changing the underlying code. Consequently, Lighter is task-agnostic and applicable to tasks ranging from classification to self-supervised learning. For example, you can implement the following criterion adapter to apply sigmoid activation to predictions and route the data to a criterion's respective arguments:

```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
```

Project-specific modules

Lighter's modular design lets researchers add custom components in organized project directories. For example, a project folder like:

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

is imported as a module named `project`, with its components accessible in configuration:

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

Research Contributions That Use Lighter

- 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 adapter design pattern idea. We thank Wenqi Li, Nic Ma, Yun Liu, and Eric Kerfoot for their continuous support with 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