

Lighter: Configuration-Driven Deep Learning

Ibrahim Hadzic^{1,2}, Suraj Pai^{2,3,4}, Keno Bressem^{5,6}, Borek Foldyna¹, 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](https://doi.org/10.xxxxxx/draft)

Software

- Review
- Repository
- Archive

Editor: [Open Journals](#)

Reviewers:

- @openjournals

Submitted: 01 January 1970

Published: unpublished

License

Authors of papers retain copyright and release the work under a

Creative Commons Attribution 4.0 International License ([CC BY 4.0](#)).

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:

- 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.*
- 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.*
- 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.*
- 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.*
- 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.

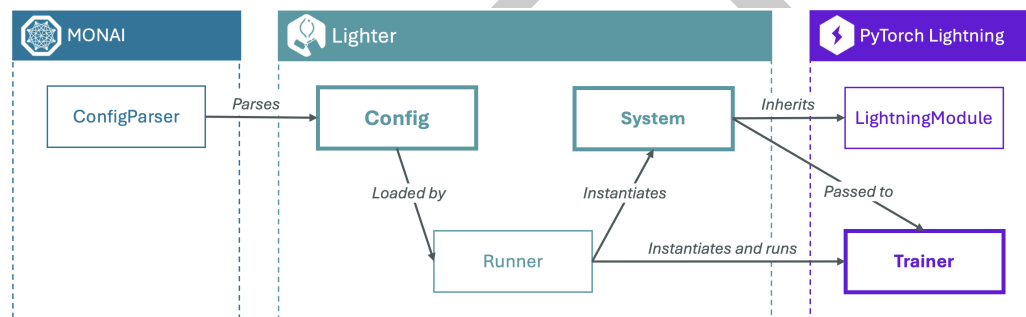


Figure 1: Lighter Overview. Lighter revolves around three main components – Trainer, System and Config, which contains the definition for the former two. Config leverages MONAI's ConfigParser for parsing the user-defined YAML configuration files, and its features are used by Runner to instantiate the 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](#)).

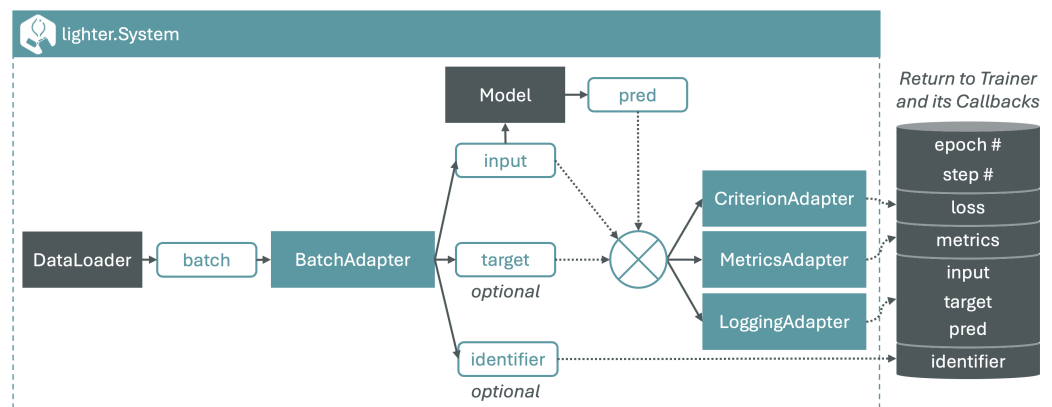


Figure 2: Flowchart of the `Lighter.System`. A batch from the `DataLoader` is processed by `BatchAdapter` to extract input, target (optional), and identifier (optional). The `Model` generates `pred` (predictions) from the input. `CriterionAdapter` and `MetricsAdapter` compute loss and metrics, respectively, 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

Adapters

The adapter pattern creates an interface layer between core system components, allowing customized data flow across the system. This flexibility enables the framework to support diverse tasks from classification to self-supervised learning. Researchers can modify component interactions without changing framework code—simply by configuring adapters. For example, applying sigmoid activation and routing predictions to the right criterion arguments can be done through criterion adapter configuration:

```

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 enables custom implementations through its modular structure. Researchers can integrate specialized components (architectures, datasets, metrics, transforms) in organized project directories, maintaining clear separation between framework functionality and project code.

For example, a project folder `joss_project`:

```

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

```

80 is imported as the project module, with components accessible in configuration:

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

81 Research Contributions That Use Lighter

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

84 Acknowledgments

85 We thank John Zielke for the adapter design pattern idea. We thank Wenqi Li, Nic Ma, Yun
86 Liu, and Eric Kerfoot for their continuous support with MONAI Bundle.

87 References

- 88 Cardoso, M. J., Li, W., Brown, R., Ma, N., Kerfoot, E., Wang, Y., Murray, B., Myronenko, A.,
89 Zhao, C., Yang, D., Nath, V., He, Y., Xu, Z., Hatamizadeh, A., Zhu, W., Liu, Y., Zheng,
90 M., Tang, Y., Yang, I., ... Feng, A. (2022). *MONAI: An open-source framework for deep*
91 *learning in healthcare*. <https://doi.org/10.48550/arXiv.2211.02701>
- 92 Falcon, W., & The PyTorch Lightning team. (2019). *PyTorch Lightning* (Version 1.4).
93 <https://doi.org/10.5281/zenodo.3828935>
- 94 Pai, S., Bontempi, D., Hadzic, I., Prudente, V., Sokač, M., Chaunzwa, T. L., Bernatz, S.,
95 Hosny, A., Mak, R. H., Birkbak, N. J., & Aerts, H. J. W. L. (2024). Foundation model for
96 cancer imaging biomarkers. *Nat. Mach. Intell.*, 6(3), 354–367. <https://doi.org/10.1038/s42256-024-00807-9>
- 98 Pai, S., Hadzic, I., Bontempi, D., Bresssem, K., Kann, B. H., Fedorov, A., Mak, R. H.,
99 & Aerts, H. J. W. L. (2025). *Vision foundation models for computed tomography*.
100 <https://doi.org/10.48550/arXiv.2501.09001>