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**Published:** unpublished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 a configuration-driven deep learning (DL) framework that separates experimental setup from code implementation. Models, datasets, and other components are defined through structured configuration files (configs). Configs serve as snapshots of the experiments, enhancing reproducibility while eliminating unstructured and repetitive scripts. Lighter uses (i) PyTorch Lightning (Falcon & The PyTorch Lightning team, 2019) to implement a task-agnostic DL logic, and (ii) MONAI Bundle configuration (Cardoso et al., 2022) to manage experiments using YAML configs.

## Statement of Need

Lighter addresses several challenges in DL experimentation:

1. **Repetitive and Error-Prone Setups**: DL typically involves significant boilerplate code for training loops, data loading, and metric calculations. The numerous hyperparameters and components across experiments can easily become complex and error-prone. Lighter abstracts these repetitive tasks and uses centralized configs for a clear, manageable experimental setup, reducing tedium and potential for errors.

2. **Reproducibility and Collaboration**: Inconsistent or complex codebases hinder collaboration and experiment reproduction. Lighter's self-documenting configs offer clear, structured snapshots of each experiment. This greatly improves reproducibility and simplifies how teams share and reuse setups.

3. **Pace of Research Iteration**: The cumulative effect of these challenges inherently slows down the research cycle. Lighter streamlines the entire experimental process, allowing researchers to focus on core hypotheses and iterate on ideas efficiently.

## State of the Field

Config-driven frameworks like Ludwig (Molino et al., 2019), Quadra (Mammana et al., 2025), and GaNDLF (Pati et al., 2023) provide high levels of abstraction by encapsulating all components within predefined structures. While this approach simplifies usage, it limits flexibility to modify the flow or extend components, often requiring direct source code changes.

Lighter takes a different approach by providing medium-level abstraction. It implements a unified flow while maintaining direct compatibility with standard PyTorch components (models, datasets, optimizers). The flow itself is modifiable to any task via adapters, while custom code is importable via config without source code modifications.

# Design

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

1. **`Config`**: serves as the experiment's blueprint, parsing and validating YAML configs that define all aspects of the experimental setup. Within these configs, 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`.

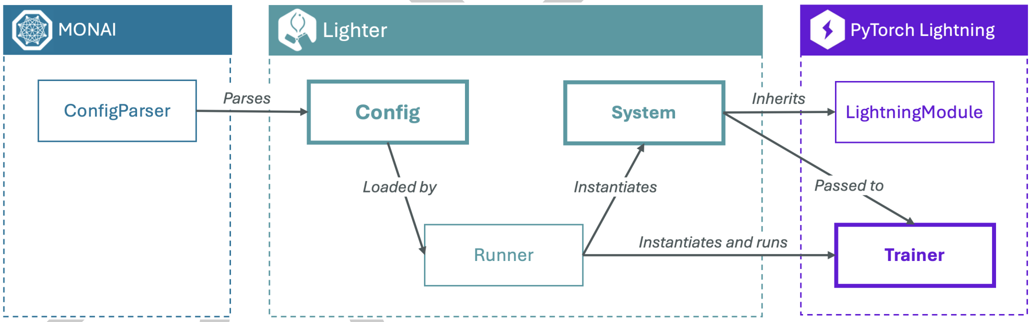


**Figure 1: Lighter Overview.** `Config` leverages MONAI's `ConfigParser` for parsing the user-defined YAML configs, and its features are used by Runner to instantiate the `System` and `Trainer`. `Trainer` is used directly from PyTorch Lightning, whereas `System` inherits from `LightningModule`, ensuring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

The Journal of Open Source Software

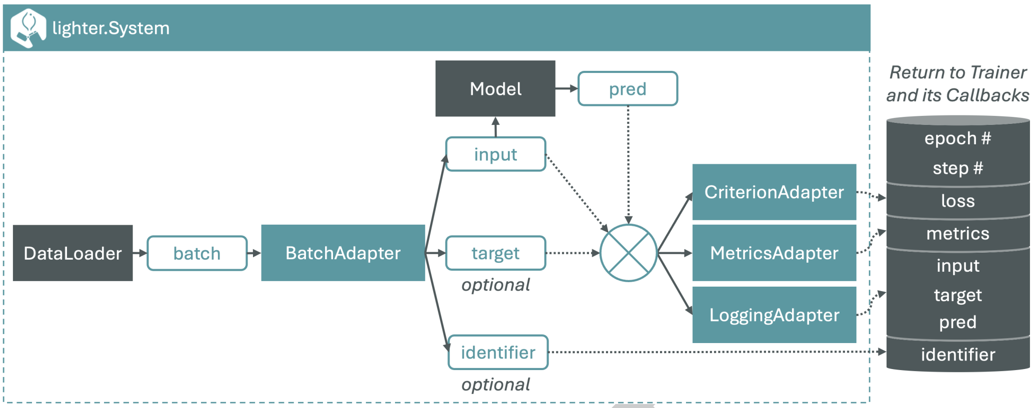


**Figure 2: Flowchart of the `lighter.System`.** A `batch` from the `DataLoader` is processed by `BatchAdapter` to extract `input`, `target` (optional), and `identifier` (optional). The `Model` generates `pred` (predictions) from the `input`. `CriterionAdapter` and `MetricsAdapter` compute loss and metrics,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

If we consider all possible DL tasks, we will find it challenging to implement a single flow that supports all. Instead, frameworks often implement per-task flows (e.g., segmentation, classification, etc.). Lighter, however, implements a unified flow modifiable via *adapter classes*. In software design, *adapter design pattern* enables components with incompatible interfaces to work together by *bridging* them using an adapter class. In Lighter, these bridges (Figure 2) specify how, for example, the model's predictions and other data are routed to the loss function or metrics. They additionally allow transformations to be applied to the data before passing it to the next component. This can be useful for tasks like binary classification, where the model's output needs to be transformed (e.g., applying a sigmoid activation function) before computing the loss or metrics. Another example would be logging, where the data often needs to be transformed before it is logged.

```yaml
# Example of an adapter transforming and routing data to the loss function
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 first arg
            target_argument: 1 # Pass 'target' to criterion's second arg
```

### Project-specific modules

Using custom components does not require modifying the framework. Instead, they can be defined within a *project folder* like:

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

By specifying the project path in the config, it is imported as a module whose components can be referenced in the config:

```yaml
project: /path/to/joss_project  # Path to the directory above
system:
    model:
        _target_: project.models.mlp.MLP  # Reference to the custom model
        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

Mammana, L., Malli, R. C., Polidori, A., & ebernatene. (2025). *Orobix/quadra*. https://github.com/orobix/quadra

Molino, P., Dudin, Y., & Miryala, S. S. (2019). Ludwig: A type-based declarative deep learning toolbox. *CoRR*, *abs/1909.07930*. http://arxiv.org/abs/1909.07930

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

Pati, S., Thakur, S. P., Hamamcı, İ. E., Baid, U., Baheti, B., Bhalerao, M., Güley, O., Mouchtaris, S., Lang, D., Thermos, S., Gotkowski, K., González, C., Grenko, C., Getka, A., Edwards, B., Sheller, M., Wu, J., Karkada, D., Panchumarthy, R., … Bakas, S. (2023). GaNDLF: The generally nuanced deep learning framework for scalable end-to-end clinical workflows. *Communications Engineering*, *2*(1), 23. https://doi.org/10.1038/s44172-023-00066-3