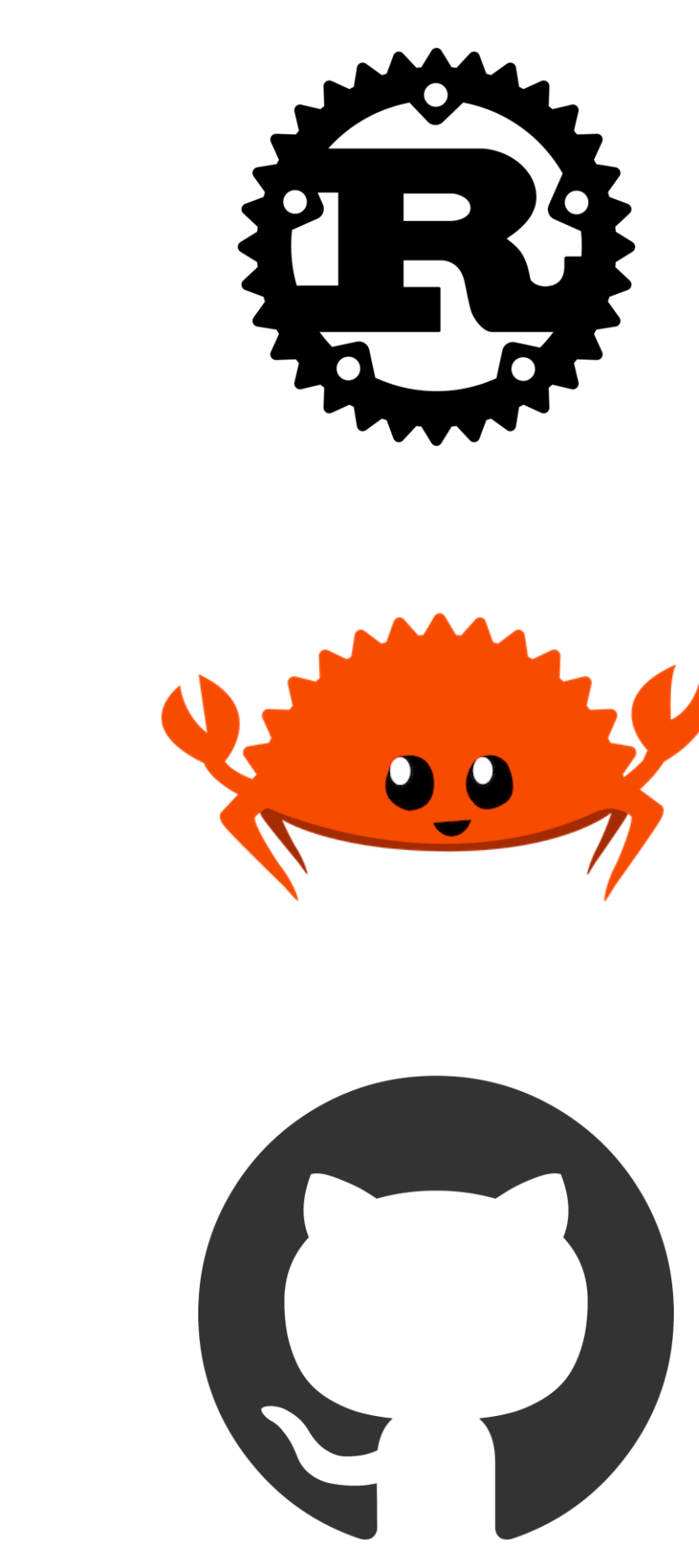


Deep Learning Framework in Rust

Student: Julio Rego
Instructor/Faculty: Dr. Masoud Sadjadi, Florida International University



PROBLEM

The main problem addressed by our project is the need for efficient and scalable deep learning frameworks that can handle large-scale datasets and models. My work focused on developing several crucial features that are essential for any deep learning framework, including model parameter loading, learning rate scheduling, and optimization algorithms. These features are critical for improving the performance and accuracy of deep learning models, especially in resource-intensive applications.

CURRENT SYSTEM

Our framework that was initially based on Andrej Karpathy's micrograd has gained many improvements. The system now includes several key features that are necessary for a robust and scalable deep learning framework and is more capable of handling larger and more complex models with improved performance and accuracy. These improvements are especially important for resource-intensive applications and make our deep learning framework an ideal choice for a wide range of deep learning tasks.

REQUIREMENTS

- Extensible architecture to enable customization and addition of new features.
- Compatibility with other popular deep learning frameworks like PyTorch.
- Memory safety.
- Capable of training complex models.
- Simple to learn and use.

CONTRIBUTIONS

- **Model parameter loading:** Enables loading of previously trained model parameters for continuing training from a previously saved checkpoint. (Fig. 1)
- **Learning rate schedulers:** Adds ability to use different learning rate schedules during the training process to optimize the training process and improve model performance. (Fig. 2)
- **Adam optimizer:** Added the powerful and flexible Adam optimizer to the framework for use during the training process. (Fig. 3)

SCREENSHOTS

```
fn load_state_dict(&mut self, path: &str) {
    let file = File::open(path).unwrap();
    let state_dict: IndexMap<String, Vec<f64>> =
        de::from_reader(file, DeOptions::new()).unwrap();

    let trainable_layers = self.layers.iter().filter(|x| x.is_trainable());

    for layer in trainable_layers {
        let (weight_key, bias_key) = (layer.name() + ".weight", layer.name() + ".bias");

        if state_dict.get(&weight_key).unwrap().len() != layer.weights().len() {
            panic!("Wrong loaded weight count for layer '{}'.", layer.name());
        }
        if state_dict.get(&bias_key).unwrap().len() != layer.biases().len() {
            panic!("Wrong loaded bias count for layer '{}'.", layer.name());
        }

        layer.set_weights(state_dict.get(&weight_key).unwrap());
        layer.set_biases(state_dict.get(&bias_key).unwrap());
    }
}
```

Figure 3: Load state_dict function

```
pub trait Schedule {
    const HAS_CLOSED_FORM: bool = true;

    fn get_lr(&self, base_lr: f64, last_epoch: usize) -> f64;
    fn get_closed_form_lr(&self, _base_lr: f64, _last_epoch: usize) -> f64 {
        0.0
    }
}

1 implementation
pub struct LRScheduler<S: Schedule> {
    lr: Value,
    schedule: S,
    base_lr: f64,
    last_epoch: usize,
}
```

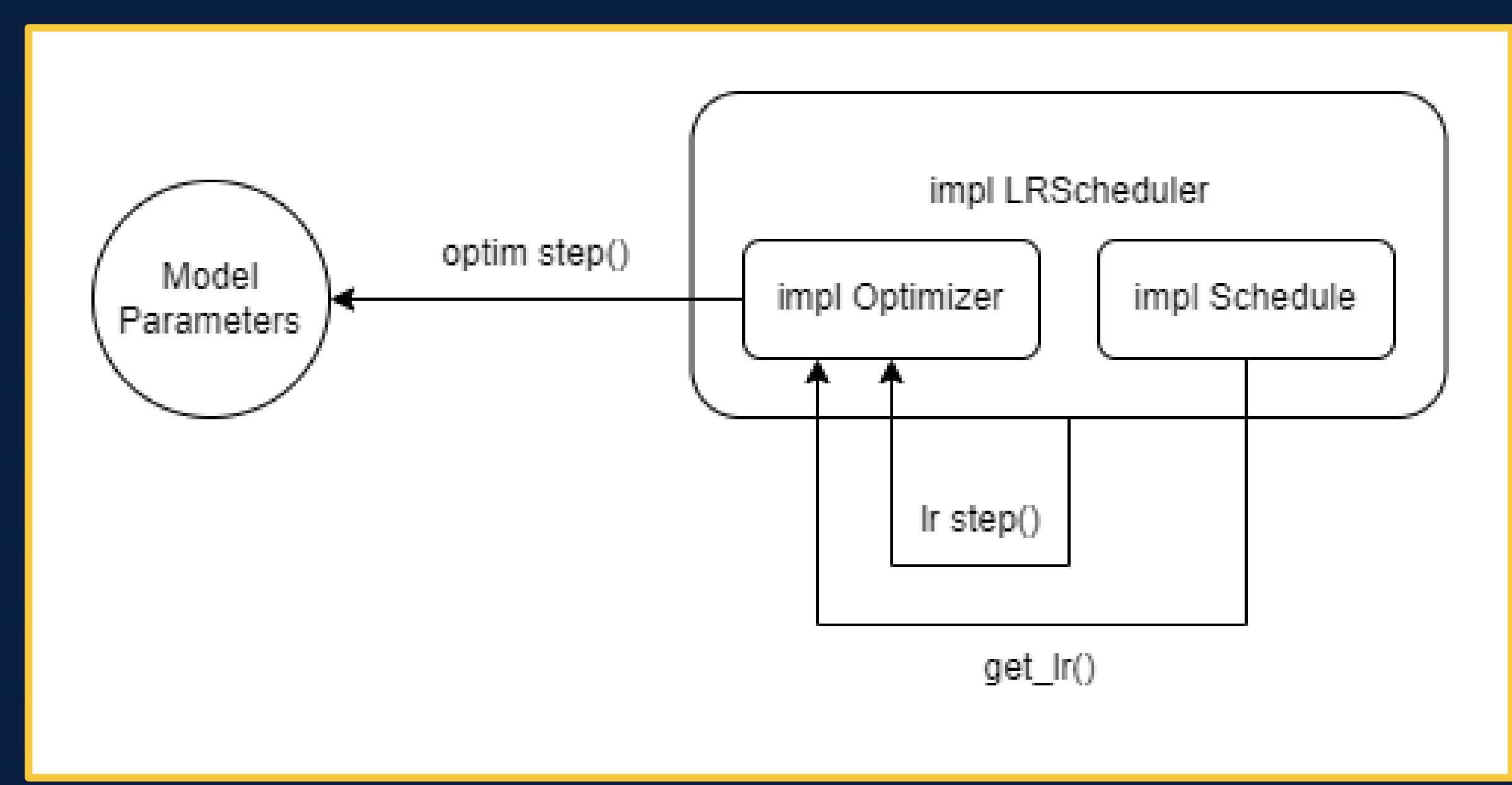
Figure 2: Learning rate scheduler structs

```
pub struct AdamCache {
    pub time_step: usize,
    pub exp_avg: Option<Array1<f64>>,
    pub exp_avg_sq: Option<Array1<f64>>,
    pub max_exp_avg_sq: Option<Array1<f64>>,
}

2 implementations
pub struct Adam {
    pub params: Vec<Value>,
    pub lr: Value,
    pub betas: (f64, f64),
    pub eps: f64,
    pub weight_decay: f64,
    pub amsgrad: bool,
    pub maximize: bool,
    pub cache: AdamCache,
}
```

Figure 3: Adam optimizer struct

SYSTEM DESIGN



SUMMARY

Developing a deep learning framework in Rust is an important project as it brings together the benefits of two powerful technologies. Rust's strong memory safety guarantees and performance make it an excellent candidate for building high-performance and reliable deep learning systems. This project has the potential to significantly enhance the existing deep learning ecosystem and pave the way for new applications of deep learning in domains that demand high reliability and safety.

REFERENCES

- Andrej Karpathy's micrograd: <https://github.com/karpathy/micrograd>
- micrograd-rs: <https://github.com/LazaroHurtado/micrograd-rs>
- PyTorch documentation on optimizers and learning rate schedulers: <https://pytorch.org/docs/stable/optim.html>