

The fault in our traits & the borrow checker

Aman Sharma

Aftershoot Inc.

January 14, 2026

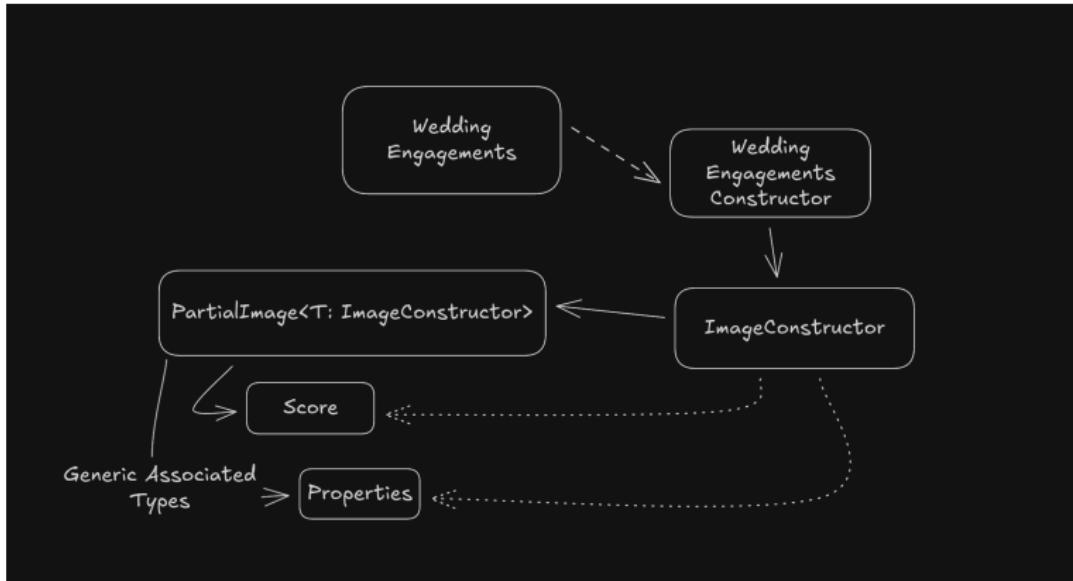
Rust was created as a replacement of CPP.

OOP is "Compile-time hierarchy of encapsulation that matches the domain model" - Casey Muratori

Now I was trying to create an encapsulation of domain model into traits.

Culling is selecting of ?good? images. The definition of good/bad is judged by multiple factor, but for the sake of this presentation we take every image, then run ML models on them get scores, and run algorithm on them to get selection/rejections.

a smallest & wrong diagram of our code structure looks like this, remind you this is of only one pipeline, each pipeline has a seperate structure like this, this causes massive code duplication.



This in term made very brittle and hard to reason about, any change and addition to the feature or code required writing something horrendous like this, or simply use a dozen of macros and work in together.

```
#[pilot]
impl<ICnstr, F> Pilot for FaceDupLayerOptIn<ICnstr, F>
where
    ICnstr: PartialImageConstructor,
    F: Fn(&PartialImage<ICnstr>) → ndarray::Array1<f32>,
    <ICnstr as PartialFaceConstructor>::PartialFaceScores:
        FaceScoresFaceDupFuxx + FaceScoresFuxx + FaceScoresImportanceChangeFuxx,
    <ICnstr as PartialImageConstructor>::PartialImageProperties: ImagePropsFuxxFaceDup,
    <ICnstr as PartialImageConstructor>::PartialImageScores:
        ImageScoresFuxxFaceDup + ImageScoresFuxxPerson,
{
    type Input<'i> = Vec<Vec<PartialImage<ICnstr>>>;
    type Output<'o> = Vec<Vec<PartialImage<ICnstr>>>;
}
```

Even with the macros, composing features was difficult

The core reason of creating any type of encapsulation is to have less code duplication, traits are good for code generation.

But they force you to think in term of little small pieces of data/methods, that can be accessed from within the traits

What's the solution??

We can flip the encapsulation boundary