Explorando Padrões de Projetos em Sistemas baseados em Aprendizado de Máquina: Um estudo de Caso

Vitor Gabriel Balsanello[1], Francisco Carlos Souza[2], Alinne Souza[3],

Universidade Tecnológica Federal do Paraná (UTFPR) - Dois Vizinhos, PR - Brasil

vitorbalsanello@alunos.utfpr[1],franciscosouza@utfpr.edu.br[2]

alinnesouza@utfpr.edu.br[3]

Abstract. *Machine Learning has become an area increasingly studied and analyzed by both software professionals and academics. The growth of projects and the specific characteristics of these products require a deeper and more comprehensive analysis from an architectural perspective. This study aims to develop an intelligent system and, at the same time, perform an analysis of software design patterns through their application in the said system. As a result, some observations were presented on the importance of these patterns and also the great need to apply them in real projects.*

Resumo. *Aprendizado de máquina tem se tornado um área cada vez mais estudada e analisada tanto por profissionais de software quanto por acadêmicos. O crescimento dos projetos e as características especificas desses produtos, exige a necessidade de uma análise mais profunda e abrangente sobre uma perspectiva arquitetural. Este estudo visa desenvolver um sistema inteligente e, ao mesmo tempo, realizar uma análise dos padrões de projeto de software por meio de sua aplicação no referido sistema. Como resultado, foram apresentadas algumas observações sobre a importância desses padrões e também a grande necessidade de aplicá-los em projetos reais.*

1. Introdução

A Engenharia de Software (ES) é uma grande área da computação que abrange desde a concepção até o desenvolvimento prático de software, enfatizando a organização de elementos internos e externos. Essas decisões organizacionais desempenham um papel crucial na prevenção de falhas em projetos de software.

A importância desse aspecto organizacional no projeto é tão significativa que levou ao desenvolvimento de uma área específica chamada Arquitetura de Software (AS). Essa área é amplamente estudada por profissionais que buscam reduzir riscos durante a implementação de novos sistemas. De acordo com [Lean Bass and RickKazman 2013], a AS é definida como o conjunto de estruturas necessárias para compreender o sistema, seus componentes, suas interações e propriedades. Quando projetos de médio ou grande porte negligenciam esses conceitos, podem surgir diversos problemas, um dos mais comuns é a erosão arquitetural.

Segundo [van Gurp and Bosch 2002] a erosão arquitetural são as consequências causadas pelo desvio das decisões arquiteturais e que pode comprometer todo o sistema principalmente em projetos que apresentam um ciclo de vida longo, como é o caso de

sistemas baseados em Aprendizado de Máquina (AM). A erosão arquitetural possui causas similares para sistemas tradicionais e sistemas inteligentes (SI), tais como mudanças constantes de requisitos, falta de padrão de desenvolvimento, tecnologias obsoletas, acoplamento entre classes, dentre outras.

Contudo, considerando sistemas baseados em AM além de possuírem as causas supracitadas, também existem problemas específicos da área. Em 2020 [Valliappa Lakshmanan 2020] publicaram o livro *Machine Learning Design Patterns: Solutions to Common Challenges in Data Preparation, Model Building, and Mlops*, a obra apresenta as principais características de *Design Patterns* envolvendo AM e quais problemas eles podem sanar considerando o ciclo de vida de desenvolvimento desses tipos de projetos.

Para combater a erosão arquitetural, é fundamental investir em boas práticas de ES, como documentação, testes abrangentes, e principalmente adesão a padrões de projeto e princípios de *design* sólidos. Assim, o objetivo deste trabalho é desenvolver um Sistema Inteligente de pequeno porte aplicando padrões de projetos de software e boas práticas de programação. Além disso, descrever um estudo de caso o qual exibe as consequências e benefícios da aplicação desses padrões.

O restante do trabalho está organizado da seguinte forma: as Seções 2 e 3 apresentam os fundamentos teóricos envolvidos na elaboração do trabalho e uma descrição dos trabalhos relacionados, respectivamente; a Seção 4 e 5 a proposta e a a metodologia necessária para realização do estudo de caso. Por fim, os resultados na Seção 6; e as conclusões do trabalho na Seção 7.

2. Referencial Teórico

2.1. Desenvolvimento de Aplicações de Aprendizado de Máquina

O AM pode ser definido como um ramo da inteligência artificial que representa uma dos campos mais importantes dentro da computação moderna, somando vários avanços nos últimos anos. O processo de desenvolvimento de aplicações do tipo, segue algumas etapas bem estabelecidas, como descreve [Huyen 2022].

- Definição do Escopo do Projeto: Nessa fase inicial, estabelece-se o escopo do projeto, incluindo seus objetivos, conceitos-chave e os principais desafios que serão enfrentados.
- Engenharia de Dados: Os modelos de AM são desenvolvidos com base no comportamento de dados conhecidos. Portanto, como próximo passo, é necessário definir o conjunto de dados a ser analisado e, em seguida, delimitar, coletar e preparar esses dados.
- Desenvolvimento do modelo de AM: Após a definição do grupo de dados, é realizado a extração das características e desenvolvimento dos modelos iniciais, com o uso das mais relevantes. Esta etapa requer alta habilidade técnica da equipe, pois inclui a seleção de algoritmos, treinamento do modelo e avaliação.
- Implantação: Após a equipe treinar e avaliar o modelo, é necessário disponibilizá-lo para uso, ou seja, colocá-lo em produção. Em geral esse é o processo mais simples do *pipeline*

- Monitoramento: Após o modelo estar operacional, é essencial monitorar seu desempenho, observando variações na acurácia, seu comportamento em diferentes contextos e sua capacidade de atender a diversos requisitos.
- Análise de Negócios: No âmbito dos negócios, é fundamental avaliar se o modelo alcança os objetivos estabelecidos. Além disso, por meio dessa prática, é possível conceber novas ideias de negócios relacionadas ao projeto.

2.2. Desafios Comuns em Aprendizado de Máquina

Projetos de AM destacam a importância de boas práticas de codificação, manutenção e implantação, devido às suas características singulares. Eles demandam uma abordagem técnica distinta da maioria dos projetos de software, enfatizando três aspectos cruciais, conforme mencionado por [Ng 2018].

A flexibilidade na aquisição de dados é crucial, sistemas de aprendizado de máquina devem ser capazes de se adaptar a mudanças nos dados. Isso pode ser especialmente relevante em cenários onde onde os dados são gerados de forma contínua, como em aplicações de IoT (Internet das Coisas) ou em ambientes de negócio.

Outra questão importante, é o tempo de treinamento, muitas vezes modelos grandes podem demorar um longo tempo para serem treinados, além de exigirem um alto custo computacional. Isso pode ser significativo quando se trata de implementar os modelos em produção, em ambientes que possuem algum tipo de restrição em tempo real.

O tempo de treinamento não afeta apenas o desenvolvimento inicial, mas também o processo de manutenção, possíveis ajustes e melhorias, já que treinar novamente modelos complexos pode ser um atividade demorada. Nesse contexto existe a necessidade das equipe considerarem cuidadosamente eficiência do treinamento, a escalabilidade e a otimização de algoritmos.

O monitoramento em produção frequentemente revela desafios na detecção e correção da queda de precisão. A adaptação a diferentes contextos e requisitos também pode ser complexa, muitas vezes resultando em desafios imprevistos para a equipe. Manter atividades monitoria constantes são essenciais, pois a queda na precisão pode ser causada por variados fatores, incluindo alterações nos dados, no contexto ou no próprio modelo.

2.3. Design Patterns

Em arquitetura de software *Design Patterns* são uma estratégia comum para resolver problemas recorrentes, que são amplamente documentados, tanto a nível de código, quanto em aspectos conceituais. Segundo [Erich Gamma and Vlissides 2009] *Design Patterns* são caracterizados através de quatro princípios comuns.

O nome do padrão é um marcador usado para descrever um problema de *design*, suas soluções e consequências em uma palavra ou duas. Essa estratégia de nomear o padrão ajuda a aumentar o vocabulário do projeto e aumenta a abstração relacionada a prática arquitetural. O problema descreve quando é necessário aplicar o padrão, o explica e seu contexto, e pode representar um problema específico do *design*.

A solução descreve os elementos que compõem o padrão, as relações, responsabilidades e colaborações entre esses. A solução não descreve um padrão específico porque

esse funciona como um *template* que pode ser aplicado em situações diferentes. Por fim as consequências relacionadas são os resultados e a relação custo-beneficio, considerando o padrão aplicado no projeto. Elas são críticas para analisar as decisões arquiteturais tomadas, as alternativas de design e entender os custos relacionados.

Atualmente em engenharia de software *Design Patterns* são amplamente utilizados e mapeados com grandes aplicações. Principalmente em produtos de software de médio a grande porte, onde escolhas arquiteturais representam mudanças significativas no software.

2.4. Padrões de Projeto Voltados a Aprendizado de Máquina

Essa Seção busca elencar padrões de aprendizado de máquina escolhidos para o trabalho, esses foram determinados a partir de sua aplicabilidade no projeto, segundo descreve [Valliappa Lakshmanan 2020]:

Explainable Predictions: O padrão *Explainable Predictions* é uma abordagem dentro de AM que permite inserir uma camada a mais de visualização no comportamento do modelo, a partir dos resultados gerados, mostrando principalmente durante o treinamento, como as decisões foram tomadas e quais os critérios utilizados. Muito útil para a configuração de parâmetros.

Repeatable Splitting: O padrão *Repeatable Splitting* é um método poderoso para testar modelos de aprendizado de máquina, esse padrão estabelece uma estratégia, que permite que a divisão entre treinamento e teste seja completamente replicável, garantindo maior precisão na comparação dos modelos, já que esses são comparados exatamente com os mesmos dados.

Data Transform: O padrão *Data Transform* simplifica o processo de tratamento de dados para treinamento. Seu objetivo principal é converter dados brutos em uma forma adequada para análise, modelagem e tomada de decisões. Isso envolve várias operações, como limpeza, normalização, padronização e discretização. Além disso, o padrão isola o tratamento de dados, promovendo desacoplamento e facilitando testes e alterações, especialmente quando diferentes tratamentos são necessários para o modelo.

Model Versioning: O padrão *Model Versioning* aborda o gerenciamento de versões de modelos de AM. Ele envolve a rastreabilidade e documentação das versões do modelo, garantindo a capacidade de comparar o desempenho e os resultados ao longo do tempo. Esse padrão é fundamental para o controle de qualidade e colaboração entre a equipe.

3. Trabalhos Relacionados

Apesar da literatura sobre padrões de projeto em aprendizado de máquina ser recente existem alguns trabalhos que mostram avanços consideráveis. Por tanto, essa Seção objetiva apresentar alguns estudos que trouxeram informações relevantes para o atual, e que oferecem contribuições importantes sobre a temática em geral.

Um dos principais livros publicados sobre o tema, foi de autoria de [Valliappa Lakshmanan 2020], a obra aborda uma serie de padrões de projetos para AM, desde os mais conceituais, com caráter de metodologia até padrões técnicos a nível de código, o objeto de estudo foram trinta padrões de projeto que foram documentados e descritos pelos autores.

Em 2019 [Hironori Washizaki and Guéhéneue 2019] Pesquisaram e classificaram uma série de padrões aplicáveis a AM, estabelecendo quais trariam vantagens para o desenvolvimento, em questões como custo e manutenção do produto. Os padrões obedecem estrutura comum dos estudados em ES.

No ano de 2021 [Ferlitsch 2021] Publicou o livro, *Deep Learning Patterns and Practices*, a obra aborda algumas alternativas de *Design Patterns* para AM e também boas práticas no desenvolvimento de projetos do tipo. O livro ainda comenta alguns fundamentos básicos sobre os temas descritos.

Partindo de uma visão contrária [Nikil Muralidhar and Ramakrishnan 2021] descreveram os principais problemas envolvendo padrões para AM, esses podem sugerir sugestões de como modelar novos para os projetos. Além disso os autores buscaram oferecer algumas alternativas que aumentassem o nível de maturidade em projetos envolvendo MLops.

No ano de 2022 [Serban and Visser 2022] descreveram adaptações arquiteturais para AM baseados em padrões comuns utilizados em ES. Os autores também conduziram um revisão de literatura acerca de arquitetura de software para AM, além de elencar os principais problemas e soluções encontrados na área.

Também descrevendo padrões arquiteturais para AM [Hironori Washizaki 2020] conduziram uma pesquisa elencando quais padrões são mais usados em projetos, conectando tanto as ES quanto AM.

A Tabela 1, disponibiliza um comparativo entre os trabalhos relacionados e o atual.

Tabela 1. Comparativo

Estudos	Objetivos
[Sharma and Davuluri 2019]	Pesquisa sobre padrões arquiteturais
[Ferlitsch 2021]	Documentação e sugestão de padrões
[Serban and Visser 2022]	Adaptação de padrões
[Nikil Muralidhar and Ramakrishnan 2021]	Sumarização de antipatterns
[Hironori Washizaki 2020]	Adaptação de padrões
Atual	Experiência de análise e benefícios dos padrões

4. Proposta

O objetivo desse trabalho é desenvolver um SI baseado em AM, com o apoio de boas práticas de desenvolvimento tais como conceitos de arquitetura e *Design Patterns*. Para esse fim serão utilizados padrões de projeto modernos voltados a AM como os descritos por [Valliappa Lakshmanan 2020]. Para atingir o objetivo geral do estudo foram definidas algumas etapas:

O modelo escolhido foi extraído da plataforma *Kaggle*, [Ulucan et al. 2019] e tem o objetivo de classificar a qualidade de carne, através de imagens submetidas. Quando uma nova imagem é enviada ao modelo, ela pode classifica-lá como ”Fresca”ou ”Estragada”.

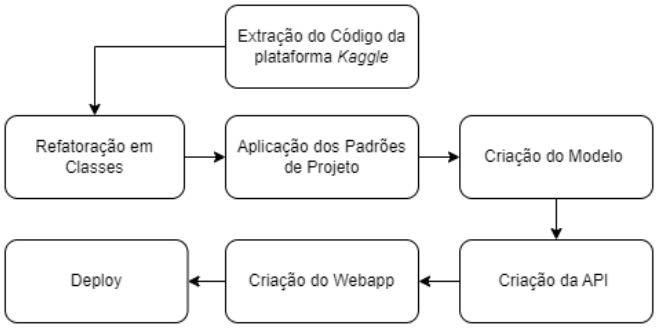


Figura 1. Metodologia Aplicada

O código inicialmente se mantinha linear em formato de *notebook*. Esse tipo de estrutura é muito útil para a geração rápido do modelo e fins didáticos mas não apresenta vantagens no que diz respeito a aspectos organizacionais.

5. Estudo de Caso

5.1. Definição e Design do Estudo de caso

No trabalho atual foi conduzido um estudo holístico de caso único como define [Yin 2014] com o objetivo de analisar os benefícios, e consequências negativas da aplicação de padrões de projeto em um SI. Nesse contexto a metodologia definida por [Basili and Weiss 1984], foi utilizada para definir o objetivo de estudo de caso:

”Implementar padrões de projeto modernos voltados a AM em um SI experimental avaliando, benefícios, desvantagens e consequências da sua aplicação”. Para a condução dos estudo de caso foi definida uma única questão de pesquisa:

- QP1: O quão os padrões de projetos AM auxiliam na qualidade de desenvolvimento de SI?

5.2. Design do Experimento

Para a realização do experimento foram aplicados quatro padrões arquiteturais modernos definidos por [Valliappa Lakshmanan 2020], esses já citados na seção 2. Esses padrões além de apresentarem relevância na literatura são aplicáveis ao projeto atual e podem mostrar resultados interessantes, foram aplicados, *Model Versioning*, *Data Transform*, *Repeatable Splitting*, *Explainable Predictions*.

5.3. Condução

A condução do estudo de caso foi realizada em uma única etapa, interna ao processo de desenvolvimento do sistema que consistiu na aplicação de quatro padrões arquiteturais voltados a AM. Os padrões foram implementados nas diferentes classes do código, relacionada ao ciclo de vida da aplicação. Depois desse processo foi realizada uma análise dos benefícios trazidos.

6. Resultados

6.1. Refatoração e Implantação do Modelo

Escrito na linguagem *Python* o código não apresentava um estilo arquitetural definido, nem padrões de projeto.

Um primeiro passo na condução da pesquisa foi a refatoração do código, estabelecendo métodos relacionados a cada etapa da criação do modelo, essa transformação pode ser verificada na Figura 3 e 4:

```
[ ]:
start = time.time()
image_data = []
image_target = []

for title in files:
    os.chdir(adress.format(title))
    counter = 0
    for i in data[title]:
        img = cv2.imread(i)
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        image_data.append(cv2.resize(img,(width, height)))
        image_target.append(title)
        counter += 1
        if counter == sample_size:
            break
    clear_output(wait=True)
    print("Compiled Class",title)
```

Figura 2. Código Original Extraído da plataforma Kaggle

```
def compile():
    image_data = []
    image_target = []

    for title in files:
        os.chdir(adress.format(title))
        counter = 0
        for i in data[title]:
            img = cv2.imread(i)
            img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
            image_data.append(cv2.resize(img,(width, height)))
            image_target.append(title)
            counter += 1
            if counter == sample_size:
                break
        clear_output(wait=True)
        print("Classe Compilada",title)
    return image_data, image_target
```

Figura 3. Código após refatoração inicial

Essa reformulação gerou algumas funções, como elencadas a seguir.

- *Compile*: Essa função carregava os dados de um *dataset* local, também encontrado na plataforma *Kaggle*.
- *Classification*: Essa função treinava o modelo com um rede neural do tipo CNN (*Convolutional Neural Network*), e exibia o gráfico do comportamento de cada geração da rede.
- Matriz: Essa função exibia a matriz de confusão do modelo, mostrando os acertos nas classificações, e erros.
- *Prediction*: Função chamada pela função *test* que realizava a real predição do modelo.
- *Test*: Função responsável pelo teste do modelo.
- *Visualization*: Essa função exibia uma matriz de 4x4 com os resultados da classificação de 16 imagens, na parte superior da imagem, o rótulo inicial, e na lateral esquerda o rótulo dado pelo modelo.
- *Plot*: Função exibia um gráfico de pizza com o número de imagens de cada categoria.

Após essa etapa foi implementada uma API com uma única rota, que fornecia o modelo treinando e também uma aplicação *web* com uma interface (*frontend*) simples, para que o usuário pudesse enviar as imagens para a classificação, para utilizar basta selecionar a imagem, o envio é feito automaticamente, após isso a API retorna o resultado da classificação, e também a imagem da carne. Sendo desenvolvida *Python* através do *framkework Flask*, e colocada em produção na plataforma *Render*, a interface da aplicação foi desenvolvida em HTML puro, e se encontra no seguinte link: https://meat-classification.onrender.com/

6.2. Resultados do Estudo de Caso

Essa Seção busca descrever quais foram os resultados do estudo de caso, ou seja as consequências da aplicação dos padrões no projeto.

O primeiro padrão aplicado foi o *Data Transform Pattern*, inserido na classe data_loader, visto nas Figuras 5 e 6. Os benefícios dessa aplicação estão relacionados princialmente com o desacoplamento e organização. Implementar esse padrão permitiu que vários testes nos padrões de cores pudessem ser realizados sem alterar a classe principal, no qual a transformação dos dados estava inserida. A Figura 5 mostra a classe data_loader com a função de tratamento de cores ainda interna, e a Figura 6 depois de uma nova classe que abstraí essa parte do código ser inserida.

```python
class DataLoader:
    def __init__(self, sample_size=500, width=100, height=100):
        self.sample_size = sample_size
        self.width = width
        self.height = height
        self.files = ['Fresh', 'Spoiled']
        self.adress = 'C:/Users/admin/Desktop/{}'
        self.data = {}
        for f in self.files:
            self.data[f] = []

    def compile_data(self):
        image_data = []
        image_target = []

        for title in self.files:
            os.chdir(self.adress.format(title))
            counter = 0
            for i in self.data[title]:
                img = cv2.imread(i)
                img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
                image_data.append(cv2.resize(img, (self.width, self.height)))
                image_target.append(title)
                counter += 1
                if counter == self.sample_size:
```

Figura 4. Classe data_loader

```python
class DataTransformer:
    def __init__(self, width=100, height=100):
        self.width = width
        self.height = height

    def transform_image(self, image_path):
        img = cv2.imread(image_path)
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        return cv2.resize(img, (self.width, self.height))

class DataLoader:
    def __init__(self, sample_size=500, width=100, height=100):
        self.sample_size = sample_size
        self.width = width
        self.height = height
        self.files = ['Fresh', 'Spoiled']
        self.adress = 'C:/Users/admin/Desktop/{}'
        self.data = {}
```

Figura 5. Aplicação do padrão *Data Transform*

A aplicação do segundo padrão, *Explanable Predictions Pattern* foi essencial para destacar as regiões críticas das imagens, que exerceram maior impacto nas previsões do modelo. Isso revelou a importância de introduzir variedade nos dados de entrada. Como resultado, a precisão do modelo foi significativamente aprimorada, ressaltando a necessidade de coletar mais imagens para enriquecer o conjunto de dados. Essa aplicação pode ser vista nas Figuras 7 e 8.

```python
def predict_image(self, image):
    img = cv2.resize(image, (self.width, self.height))
    test = img / 255.0
    pred = self.model.predict(np.array([image])).argmax()
    return self.labels.inverse_transform([pred])[0]

def test_images_model(self, image_data_):
    plt.figure(figsize=(15, 15))
    for i in range(1, 17):
        fig = np.random.choice(np.arange(self.size))
        plt.subplot(4, 4, i)
        plt.imshow(image_data_[fig])
        if self.image_target[fig] == 'Fresh':
            c = 'green'
        else:
            c = 'red'
        plt.title(self.image_target[fig], color=c)
        plt.ylabel("| Pred:{} |".format(self.predict_image(image_data_[fig])),
                   fontsize=17, color=c)
        plt.xticks([]), plt.yticks([])
    plt.show()
```

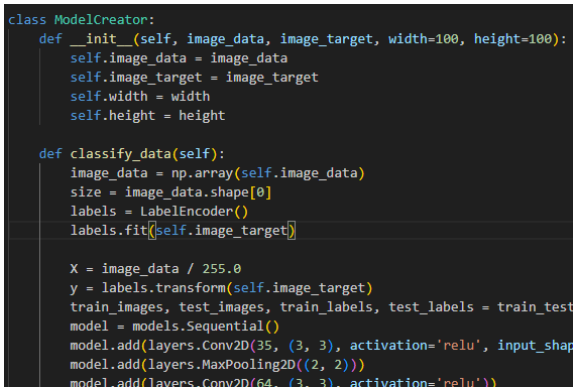
Figura 6. Classe model_tester

```python
class ExplainablePredictions:
    def __init__(self, model, width=100, height=100):
        self.model = model
        self.width = width
        self.height = height
        self.explainer = LimeImageExplainer()

    def generate_explanation(self, image_data, model_prediction):
        explanation = self.explainer.explain_instance(
            image_data[0].astype('double'),
            self.model.predict,
            top_labels=1,
            hide_color=0,
            num_samples=1000
        )
        explanation.show_in_notebook()
```

Figura 7. Aplicação do Padrão *Explainable Predictions*

O terceiro padrão, *Model Versioning Pattern* permitiu que existisse um método de versionamento a nível de código da aplicação, foi benéfico, pois em um ambiente onde várias e versões são geradas em pouco tempo, não é prático documentar as alterações de forma tão criteriosa, utilizando uma ferramenta como *Git* por exemplo. A Figura 9 mostra a classe de criação do modelo, e a 10, o código de implementação do modelo abstraído e as estratégias para guardar os diferentes modelos gerados.

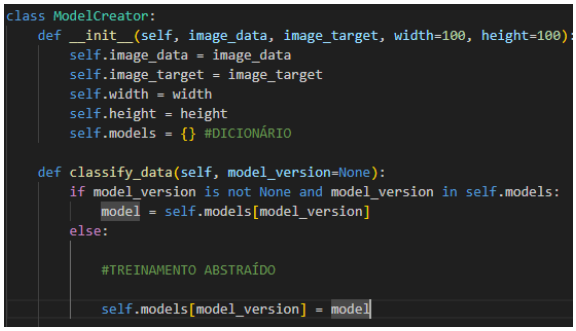


```
class ModelCreator:
    def __init__(self, image_data, image_target, width=100, height=100):
        self.image_data = image_data
        self.image_target = image_target
        self.width = width
        self.height = height

    def classify_data(self):
        image_data = np.array(self.image_data)
        size = image_data.shape[0]
        labels = LabelEncoder()
        labels.fit(self.image_target)

        X = image_data / 255.0
        y = labels.transform(self.image_target)
        train_images, test_images, train_labels, test_labels = train_test
        model = models.Sequential()
        model.add(layers.Conv2D(35, (3, 3), activation='relu', input_shap
        model.add(layers.MaxPooling2D((2, 2)))
        model.add(layers.Conv2D(64, (3, 3), activation='relu'))
```

Figura 8. Classe model_tester



```
class ModelCreator:
    def __init__(self, image_data, image_target, width=100, height=100):
        self.image_data = image_data
        self.image_target = image_target
        self.width = width
        self.height = height
        self.models = {} #DICIONÁRIO

    def classify_data(self, model_version=None):
        if model_version is not None and model_version in self.models:
            model = self.models[model_version]
        else:

            #TREINAMENTO ABSTRAÍDO

            self.models[model_version] = model
```

Figura 9. Aplicação do padrão *Model Versioning*

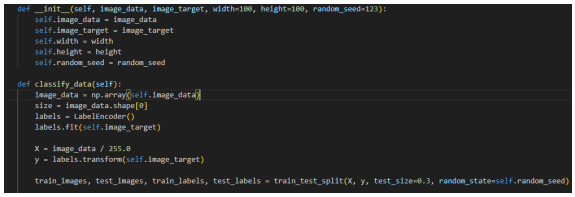
O último padrão, *Repeatable Splitting Pattern* mostrado nas Figuras 11 e 12, foi útil, pois com ele foi possível repetir os testes realizados por todos os participantes do projeto, garantindo a confiabilidade dos testes e da avaliação. Esse padrão apesar de simples consegue abstrair a definição da *seed* da classe principal, excluindo a necessidade de alteração da classe de treinamento, que é a mais importante nesse contexto, evitando possíveis erros referentes a manipulação dessa classe.

```
class ModelCreator:
    def __init__(self, image_data, image_target, width=100, height=100):
        self.image_data = image_data
        self.image_target = image_target
        self.width = width
        self.height = height

    def classify_data(self):
        image_data = np.array(self.image_data)
        size = image_data.shape[0]
        labels = LabelEncoder()
        labels.fit(self.image_target)

        X = image_data / 255.0
        y = labels.transform(self.image_target)
        train_images, test_images, train_labels, test_labels = train_test_split(X, y, test_size=0.3, random_state=123)
        model = models.Sequential()
        model.add(layers.Conv2D(35, (3, 3), activation='relu', input_shape=(self.width, self.height, 3)))
```

Figura 10. Classe model_creator



```
def __init__(self, image_data, image_target, width=100, height=100, random_seed=123):
    self.image_data = image_data
    self.image_target = image_target
    self.width = width
    self.height = height
    self.random_seed = random_seed

def classify_data(self):
    image_data = np.array(self.image_data)
    size = image_data.shape[0]
    labels = LabelEncoder()
    labels.fit(self.image_target)

    X = image_data / 255.0
    y = labels.transform(self.image_target)

    train_images, test_images, train_labels, test_labels = train_test_split(X, y, test_size=0.3, random_state=self.random_seed)
```

Figura 11. Aplicação do *Repeatable Splitting Pattern*

Portanto, como resposta a QP1, os padrões de projeto de AM utilizados neste artigo desempenharam um papel crucial na melhoria da qualidade do desenvolvimento do SI, proporcionando maior organização, adaptabilidade e confiabilidade. Foi observado que o uso dos padrões trouxe ao sistema mais facilidade de manutenção/evolução e também redução de risco ao alterar algum módulo do mesmo.

7. Conclusão

Hoje em dia existe uma necessidade muito grande de se estabelecer um literatura confiável sobre AM principalmente no que tange a *Design Patterns*, esse tipo de análise é necessária devido aos problemas específicos que projetos do gênero podem enfrentar. Sendo assim,

este trabalho buscou apresentar um projeto que pudesse mostrar estratégias relevantes sobre o tema e trouxessem alguma experiência aplicada sobre sistemas de pequeno porte.

Em um contexto amplo, a complexidade desses projetos, o desenvolvimento difícil, custoso, e cada vez mais intolerante a erros, adiciona ainda mais variáveis quando se trata da implantação da aplicação. Por fim, o trabalho conseguiu apresentar o uso de padrões de projetos associados a atividades do ciclo de desenvolvimento de sistemas baseados em AM. A partir das experiências abordadas é possível entender como esses padrões podem ser benéficos em projetos futuros e como essas análises podem ser importantes para a literatura.

Referências

Basili, V. R. and Weiss, D. M. (1984). A methodology for collecting valid software engineering data. pages 728 –738.

Erich Gamma, Richard Helm, R. J. and Vlissides, J. (2009). *Design Patterns: Elements of Reusable Object-Oriented Software*. 1st edition.

Ferlitsch, A. (2021). *Deep Learning Patterns and Practices*. 1st edition.

Hironori Washizaki, Hiromu Uchida, F. K. and Guéhéneue, Y.-G. (2019). Studying software engineering patterns for designing machine learning systems. pages 49–54.

Hironori Washizaki, Hiromu Uchida, F. K. Y.-G. G. (2020). Machine learning architecture and design patterns. pages 1–8.

Huyen, C. (2022). *Design Machine Learning Systems*. 1st edition.

Lean Bass, P. C. and RickKazman (2013). *Software Architecture in Practice*. 3rd edition.

Ng, A. (2018). *Machine Learning Yearing*. 1st edition.

Nikil Muralidhar, Sathappan Muthiah, P. B. M. J. Y. Y. K. B. W. L. D. J. P. A. H. S. M. and Ramakrishnan, N. (2021). Using antipatterns to avoid mlops mistakes. pages 1–9.

Serban, A. and Visser, J. (2022). Adapting software architectures to machine learning challenges. pages 152–163.

Sharma, R. and Davuluri, K. (2019). Design patterns for machine learning applications. pages 818–821.

Ulucan, O., Karakaya, D., and Turkan, M. (2019). Meat quality assessment based on deep learning. In *2019 Innovations in Intelligent Systems and Applications Conference (ASYU)*, pages 1–5. IEEE.

Valliappa Lakshmanan, Sara Robinson, M. M. (2020). *Machine Learning Design Patterns: Solutions to Common Challenges in Data Preparation, Model Building, and Mlops*. 1st edition.

van Gurp, J. and Bosch, J. (2002). Design erosion: problems and causes. pages 1–15.

Yin, R. K. (2014). *Case Study Research*. 5th edition.