Test à base de scénarios de programmes apprenant en ligne

Maxence Demougeot, Sylvie Trouilhet, Jean-Paul Arcangeli, Françoise Adreit

Test à base de scénarios de programmes apprenant en ligne

Maxence Demougeot[1], Sylvie Trouilhet[1], Jean-Paul Arcangeli[1], Françoise Adreit[2]

[1] IRIT, Université de Toulouse, UT3, Toulouse, France
[2] IRIT, Université de Toulouse, UT2J, Toulouse, France

{Prénom.Nom}@irit.fr

Résumé

En apprentissage automatique, un modèle est un programme qui fait des prédictions ou prend des décisions. Il ne résulte pas d'une activité de programmation traditionnelle mais d'une construction automatique par un programme apprenant alimenté par des exemples. Comme tout programme, les programmes apprenants doivent être vérifiés et validés. Le test est un moyen d'y parvenir et d'assurer une certaine confiance dans une solution d'apprentissage automatique. Dans cet article, nous proposons une analyse du problème du test logiciel dans le contexte de l'apprentissage automatique, et nous approfondissons l'état de l'art en nous concentrant sur le programme apprenant et en mettant l'accent sur l'apprentissage en ligne et interactif. Face au non-déterminisme possible, à la dynamique de l'apprentissage en ligne et à la présence de l'utilisateur dans l'apprentissage interactif, nous proposons une approche à base de scénarios pour le test de programmes apprenants. En outre, nous présentons deux outils prototypes qui permettent l'implantation et l'exécution de scénarios pour évaluer un programme apprenant en interaction avec l'utilisateur.

Mots-clés

Ingénierie des systèmes logiciels à base d'apprentissage automatique, évaluation de programmes apprenants, test, scénario, apprentissage en ligne, apprentissage interactif.

Abstract

A machine learning (ML) model serves as a program for making predictions or decisions. It is not built by traditional programming but automatically by a learning program fed by examples. As any program, learning programs need to be verified and validated. Testing is a way of doing this and providing trust in ML. In this paper, we provide an analysis of the testing problem in ML context, and go deeper into the state-of-the-art focusing on the learning program with particular emphasis on online and interactive ML (IML). In front of the possible non-determinism, the dynamics of online learning, and the presence of the user in IML, we propose a scenario-based approach to test learning programs. In addition, we present a prototype tooling that supports scenario implementation and running for evaluation purposes of a learning program that interacts with the user.

Keywords

Engineering Software Systems based on Machine Learning, Evaluation of Learning Programs, Testing, Scenario, Online Learning, Interactive Learning.

1 Introduction

Pour qu'une machine effectue une tâche, son comportement doit être programmé. Cependant, il n'est pas toujours possible ou envisageable d'écrire un programme, par exemple si l'algorithme est inconnu ou trop complexe à mettre en œuvre. Dans de tels cas, les équipes de développement peuvent opter pour des logiciels basés sur l'apprentissage automatique (*machine learning* ou ML) [17] qui visent à déduire et généraliser le comportement attendu de la machine à partir d'exemples. Comme les logiciels « traditionnels » (ceux qui ne sont pas basés sur l'apprentissage automatique), la qualité de ce type de logiciel doit être évaluée. Cette évaluation peut se faire par le test, mais ceci reste un défi majeur. La recherche dans le domaine est récente. Les travaux ciblent majoritairement l'apprentissage supervisé hors ligne et n'englobent pas tous les paradigmes d'apprentissage automatique, en particulier l'apprentissage en ligne [24] ou l'apprentissage interactif [9].

Dans cet article, nous prenons un point de vue « ingénierie logicielle » : l'objectif de notre travail est d'**accompagner le développement de solutions à base d'apprentissage automatique** au moyen de méthodes et d'outils de test qui participent à la réalisation de produits fiables qui répondent aux besoins. Le processus de développement et de production des logiciels basés sur l'apprentissage automatique est d'abord comparé à celui des logiciels traditionnels afin de mettre en évidence les défis posés par le test, en particulier lorsque l'apprentissage est fait en ligne ou lorsque l'utilisateur humain est au centre du processus. **Nous ciblons ensuite le programme apprenant**, et nous analysons la problématique du test dans le cadre de l'apprentissage en ligne et de l'apprentissage interactif. Notre approche pour la **conception**, l'**implantation** et l'**exécution** des cas de test est basée sur la notion de scénario. Nous avons conçu deux outils pour implanter et exécuter des scénarios afin d'évaluer un programme apprenant qui interagit avec l'utilisateur pour construire des applications « ambiantes ».

Ces travaux font partie du projet de recherche OppoCompo dont l'objectif est de développer une solution à base d'ap-

prentissage automatique qui construit à la volée des applications en environnement ambiant (Internet des Objets, Ville Intelligente…), avec l'utilisateur dans la boucle. Nous développons un prototype de « moteur de composition opportuniste » appelé OCE qui apprend en ligne par renforcement [27] et de manière interactive [9], les besoins et les préférences de l'utilisateur, afin de proposer des applications pertinentes en fonction du contexte [30]. La pertinence de ses résultats, l'adaptation à l'environnement ambiant et à l'utilisateur sont des qualités que nous voulons évaluer afin de vérifier que notre solution apprenante est fonctionnelle et digne de confiance.

Cet article est organisé de la manière suivante. La section 2 compare les processus de développement des logiciels traditionnels et des logiciels à base d'apprentissage automatique. La section 3 différencie le test de modèles et le test de programmes apprenants. Puis elle présente les principales difficultés du test d'un programme apprenant en ligne et avec l'utilisateur dans la boucle, en s'appuyant sur les principaux travaux du domaine. La section 4 propose le concept de scénario de test, puis présente deux outils pour implanter (Maker) et exécuter (Runner) des scénarios à des fins d'évaluation ainsi que leur intégration dans un système logiciel qui apprend en ligne en interaction avec l'utilisateur. La section 5 présente comment d'autres travaux utilisent la notion de scénario pour tester des logiciels basés sur l'apprentissage automatique et fait état de quelques outils de développement de solutions d'apprentissage par renforcement. Enfin, la section 6 résume notre contribution et ses limites, et discute quelques perspectives.

2 Développement de logiciels à base d'apprentissage automatique

Nous analysons la manière dont les logiciels basés sur l'apprentissage automatique sont développés et mis en production par rapport aux logiciels traditionnels, en se focalisant sur l'apprentissage en ligne et l'apprentissage interactif.

La figure 1 présente une vue synthétique du processus classique de développement et de production de logiciels traditionnels : à partir des exigences, l'équipe de développement met en œuvre plus ou moins manuellement une solution sous la forme d'un programme, puis le programme est exécuté et produit des résultats à partir des entrées.

La figure 2 présente une vue synthétique du processus dans le contexte de l'apprentissage hors ligne. À l'équipe de développement s'ajoutent les experts en apprentissage automatique, les experts du domaine et potentiellement les futurs utilisateurs ou leurs représentants qui collaborent pour concevoir ou sélectionner puis paramétrer le programme apprenant et définir les données d'entraînement. À l'instar du développement logiciel traditionnel, il s'agit de construire un programme, appelé **modèle**. Contrairement aux logiciels traditionnels, le modèle est construit automatiquement au cours d'une phase d'apprentissage (également appelée phase d'entraînement) à l'aide d'un **programme apprenant** alimenté par des données d'apprentissage. Les données d'apprentissage sont généralement des couples composés d'une entrée et de la sortie associée, l'ensemble des données devant être complet, cohérent et représentatif de ce que le logiciel doit apprendre [26]. Comme pour les logiciels traditionnels, une fois construit et validé, le modèle est mis en production.

2.1 Apprentissage en ligne

Il est parfois difficile d'anticiper les données que le modèle rencontrera en production, et le modèle doit s'adapter au fil du temps en fonction des données reçues. L'apprentissage automatique en ligne cible cette question. Selon S. Russel et P. Norvig [24], cette approche repose sur des comparaisons répétées entre les résultats produits et les résultats attendus, avec, éventuellement, une phase préalable d'apprentissage : lorsque le modèle produit une sortie pour une entrée donnée (décision), un expert du domaine lui fournit la bonne réponse (la sortie attendue) afin de déclencher une nouvelle phase d'apprentissage, comme le montre la figure 3. Ainsi, il y a une alternance continue entre phase d'apprentissage et phase de production ; le programme apprenant met à jour le modèle de manière itérative tout au long de son exécution.

Contrairement au cas de l'apprentissage hors ligne, le programme apprenant opère donc quand le modèle est en production pour l'adapter et le faire évoluer. Dans un contexte ouvert et possiblement non prévisible, il doit traiter des données d'apprentissage qui arrivent au fur et à mesure de l'exécution, qui peuvent ne pas être de bonne qualité mais dont les éventuels défauts ne peuvent pas être corrigés.

2.2 Apprentissage interactif

L'apprentissage automatique interactif [9] place l'utilisateur humain au centre du processus d'apprentissage [2]. Il est fréquemment combiné avec l'apprentissage en ligne. Dans ce cas, les experts en apprentissage automatique ne sont pas partie prenante dans le processus. Ce sont les utilisateurs finaux qui alimentent le programme apprenant afin de personnaliser le modèle créé pour leur propre usage, même s'ils n'ont généralement pas de compétences en apprentissage automatique. En pratique, les utilisateurs peuvent fournir un retour après chaque décision ou prédiction du modèle, ce qui permet à ce dernier de s'adapter à leurs besoins et à leurs préférences au fur et à mesure de son exécution, afin d'améliorer les prédictions futures. En contrepartie, l'apprentissage automatique interactif est sujet à la versatilité de l'utilisateur humain.

3 Test de logiciels à base d'apprentissage automatique

3.1 Principes de base du test de logiciels

Le test est une activité au sein du processus de développement des logiciels. Il permet principalement de : *(1)* mettre en évidence des situations où le programme ne se comporte pas comme attendu, afin de pouvoir apporter une correction, *(2)* montrer que le programme répond aux exigences, pour convaincre les parties prenantes que le programme fonctionne, *(3)* mettre au point le paramétrage, et *(4)* comparer différentes versions d'un même programme [3]. Les tests

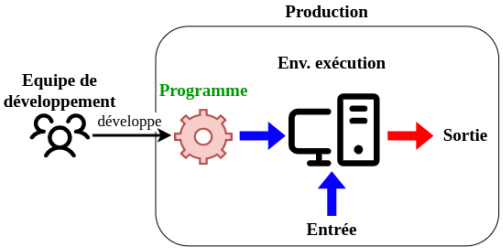

FIGURE 1 – Processus traditionnel de développement et de production de logiciels

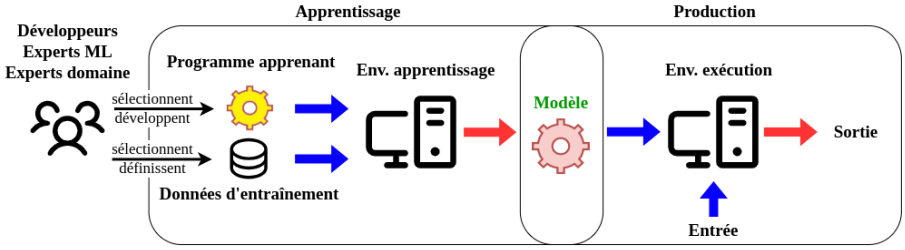

FIGURE 2 – Développement et production dans des environnements d'apprentissage hors ligne

consistent généralement à exécuter le programme dans un environnement proche de l'environnement de production, puis à analyser les résultats. Dans une équipe de développement, « l'oracle » évalue et interprète ces résultats. Si ces résultats montrent que le programme ne se comporte pas comme attendu, l'équipe de développement recherche la présence d'un défaut, reprend le développement pour le corriger, puis effectue un nouvelle série de tests. Un défaut (ou bug) est défini comme une imperfection ou une déficience dans un produit logiciel qui ne répond pas aux exigences ou aux spécifications [11]. La notion de défaut et le problème de leur détection sont analysés dans [3].

3.2 Analyse de la problématique

Contrairement aux travaux menés dans le domaine du test de logiciels traditionnels, la recherche sur le test de logiciels basés sur l'apprentissage automatique est récente. Plusieurs articles traitent du problème général du développement de logiciels basés sur l'apprentissage automatique avec un point de vue ingénierie logicielle, mais ne fournissent que de brèves explications sur les questions du test [10, 16].

Cependant, quelques articles importants ont été publiés ces dernières années. Zhang et al. [31] ont proposé une étude complète du test de logiciels basés sur l'apprentissage automatique, en mettant l'accent sur l'évaluation du modèle et de propriétés telles que l'exactitude, la robustesse, l'équité. Riccio et al. [23] ont analysé la littérature de manière systématique et mis en évidence les principaux défis liés au test, notamment la spécification des cas de test, les critères d'adéquation, le coût et le problème de l'oracle.

Pour analyser la problématique du test de logiciel à base d'apprentissage automatique, nous soulignons la différence entre le test de modèles et le test de programmes apprenants. Puis nous nous concentrons sur le test de l'apprentissage en ligne et de l'apprentissage interactif.

3.2.1 Test de modèles *vs* test de programmes apprenants

L'apprentissage automatique recouvre des activités d'apprentissage (construction de modèle) et de décision (exploitation du modèle), que ces activités soient entrelacées ou non. En matière de test, il faut donc distinguer deux volets : le test du modèle et le test du programme apprenant.

Tester le modèle consiste à l'exécuter pour répondre à la question suivante : le modèle est-il un « bon » modèle ? En d'autres termes, la machine a-t-elle « bien » appris ? Le test de modèles a les mêmes objectifs que le test de logiciels traditionnels mais en mettant l'accent sur des propriétés de qualité du modèle telles que l'exactitude, la pertinence ou la robustesse [31]. Cela pose plusieurs problèmes que nous examinons ci-dessous.

Comme les modèles résultent de l'exécution d'un programme apprenant alimenté par des exemples, les défauts peuvent provenir du programme apprenant, des données d'apprentissage ou d'une inadéquation entre les deux (lorsqu'un programme apprend mal sur certaines données) [31]. En pratique, mettre l'accent sur des propriétés peut aider à localiser les sources des défauts : par exemple, tester la pertinence du modèle (comme le sur-apprentissage [12]) permet de trouver des défauts dans les données d'apprentissage.

Il peut être difficile de remonter à la source d'un défaut afin de le corriger. En effet, les modèles n'ont pas la même matérialité que les logiciels traditionnels : ils ne consistent pas en un code source mais fonctionnent le plus souvent en « boîte noire » et sont composés de divers éléments plus ou moins tangibles (code, paramètres, données).

D'autre part, comme c'est aussi traditionnellement le cas, il est essentiel pour l'évaluation des modèles de sélectionner de manière appropriée les données utilisées pour le test.

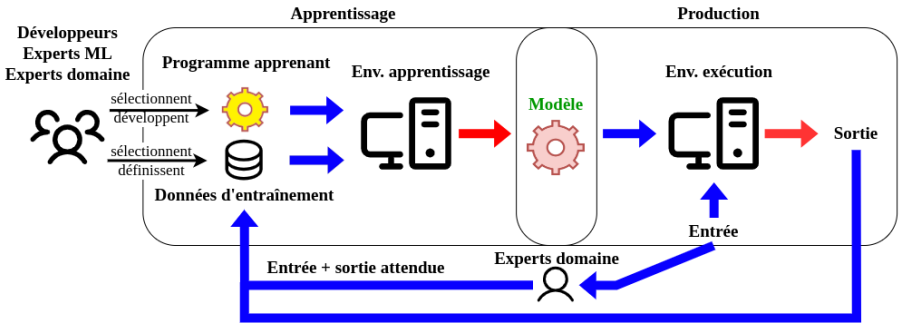

FIGURE 3 – Développement et production dans des environnements d'apprentissage en ligne

Mais, ici, l'espace est potentiellement complexe et infini.
Enfin, l'apprentissage automatique est parfois utilisé par les équipes de développement dans des situations où les résultats attendus ne sont pas connus à l'avance [18]. Dans ce cas, prédire et interpréter les résultats des tests est un défi supplémentaire, et il peut être difficile de déterminer si un test passe ou non. Les logiciels présentant ce problème, appelé le problème de l'oracle, sont souvent considérés comme non testables [29] en raison de l'absence d'oracle ou de la difficulté d'en concevoir [19].

Tester le programme apprenant n'est pas la même chose que tester un modèle, bien que les deux problèmes soient étroitement liés. La question est la suivante : le programme apprenant apprend-il « bien » dans l'ensemble du champ d'application pour lequel il a été conçu ? En d'autres termes, le programme apprenant construit-il de « bons » modèles relativement aux données qui lui sont fournies ? Pour répondre à cette question, les testeurs doivent faire en sorte que le programme apprenant construise différents modèles pour différents cas d'utilisation, dans le but de tester ensuite ces modèles. La sélection des données d'apprentissage pour construire des modèles est essentielle pour la qualité de l'ensemble des tests. En outre, comme il n'est pas possible d'exprimer les modèles attendus pour faire des comparaisons (le problème de l'oracle à nouveau), chaque modèle construit doit être exécuté pour évaluer la qualité du programme apprenant. Le test de programmes apprenants est par conséquent coûteux ; il demande une forte expertise et, autant que possible, une automatisation.
C'est le problème du test de programmes apprenants que nous ciblons ici.

Test de programmes apprenants *vs* test de compilateurs. Programmes apprenants et compilateurs sont des programmes dont l'exécution produit des programmes. Ainsi, le test de programmes apprenants présente des similitudes avec le test de compilateurs. Nous développons ici cette analogie pour aider à mieux distinguer test du modèle et test du programme apprenant.
La figure 1 cache une étape importante du développement de logiciels : la compilation. Dans le cas général, le programme mis en production est produit par un compilateur à partir d'un programme source, comme un modèle est produit à partir d'un programme apprenant.
Dans [7], les auteurs passent en revue les travaux portant sur le test de compilateurs, en mettant l'accent sur la propriété de correction, c'est-à-dire la conformité sémantique entre le programme source et le programme exécutable produit par le compilateur. Comme les programmes apprenants, les compilateurs sont des logiciels complexes dotés de multiples fonctions et paramètres. Ils manquent de spécifications précises, ce qui rend leur vérification difficile. Entre autres, le problème de l'oracle est aussi présent car il est tout à fait impossible pour les testeurs de fournir le résultat attendu (programme exécutable) à comparer avec le résultat obtenu pendant le test. Par conséquent, comme pour un programme apprenant, un compilateur ne peut être évalué qu'indirectement en testant ce qu'il produit (le programme exécutable).
De plus, pour vérifier qu'il produit systématiquement le bon exécutable, l'ensemble des entrées utilisées pour les tests (les programmes sources) doit couvrir le plus possible les différents types de programmes qui seront à compiler.

3.2.2 Test de programmes apprenant en ligne

Dans le cas de l'apprentissage en ligne, le test pose des problèmes particuliers que nous examinons ici.
De manière générale, la qualité d'un modèle dépend de la qualité des données d'apprentissage. Comme celle-ci ne peut pas être contrôlée dans le cadre de l'apprentissage en ligne, la question du test ne porte plus vraiment sur la présence d'un défaut dans les données d'apprentissage. Il s'agit plutôt de vérifier la cohérence entre les données d'apprentissage et le modèle construit. On peut aussi chercher à vérifier la robustesse, c'est-à-dire la capacité de la solution à opérer convenablement lorsque les données d'apprentissage sont de mauvaise qualité.
Pour tester, il faut imaginer un ensemble de cas que la solution pourra rencontrer en production. Or, il n'est pas toujours possible de savoir à l'avance à quelles données le programme apprenant et le modèle seront confrontés. Il est donc difficile de définir *a priori* des cas de tests représentatifs des futures données réelles et, pour le testeur, d'anticiper certains cas qui pourraient se présenter.
D'autre part, les cas de test doivent non seulement permettre l'évaluation à proprement parler mais aussi construire préalablement le modèle, ce qui complexifie à la fois la conception et l'exécution du test.
De plus, les modèles construits par apprentissage en ligne

sont soumis de manière continue à des changements (si l'apprentissage ne converge pas ou si l'environnement n'est pas stationnaire). Autrement dit, en production, ils évoluent. Les tester à un certain stade peut n'avoir que peu voire pas de sens, car ils peuvent devenir rapidement obsolètes. Parce que le processus d'apprentissage en ligne est itératif, le choix du « bon » moment pour tester se pose, et il est difficile de décider quand un modèle est suffisamment mature pour être testé.

Un autre problème réside dans la nature possiblement non-déterministe de l'apprentissage automatique [15, 26], notamment (mais pas seulement) dans le cas de l'apprentissage en ligne. La part d'aléatoire présente dans les mécanismes d'apprentissage et de décision conduit à des résultats variables d'une exécution à l'autre, avec la même configuration et les mêmes entrées. Ainsi, il peut arriver que le modèle ne produise pas les résultats attendus lors des tests alors que le mécanisme d'apprentissage fonctionne correctement [14]. Par exemple, dans l'apprentissage par renforcement (généralement effectué en ligne), il est normal que la machine choisisse parfois, à des fins d'exploration, une solution qui n'est ni la meilleure ni celle logiquement attendue. Il est donc difficile de déterminer si une sortie inattendue résulte d'un facteur aléatoire ou d'un défaut dans le mécanisme d'apprentissage.

3.2.3 Test de programmes apprenants en interaction avec l'humain

La présence d'un utilisateur humain dans la boucle complexifie la conception et la réalisation des tests.

D'une part, les résultats produits par un modèle peuvent convenir à un utilisateur mais pas à un autre. D'autre part, les besoins, les attentes ou les préférences des utilisateurs peuvent varier dans le temps et en fonction de leur situation. Il est donc difficile de créer des cas de test qui englobent un large éventail de profils d'utilisateurs tout en anticipant leur dynamique ou leurs possibles incohérences.

Un autre problème réside dans la qualité des données fournies par l'utilisateur lorsqu'il interagit avec le modèle. Par exemple, il peut mal comprendre les résultats du modèle et donner un retour non pertinent. Dans ce cas, le modèle risque de produire des résultats de test incorrects alors que le mécanisme d'apprentissage fonctionne correctement. Par conséquent, il est difficile de déterminer si la machine n'a pas réussi à apprendre ou si le problème provient des interactions entre l'utilisateur et la machine apprenante.

3.2.4 Synthèse

Nous formulons ci-dessous les questions de recherche que nous avons identifiées concernant le test de programmes apprenant de manière automatique. Les 3 premières sont relatives à la conception et à l'implantation des cas de tests, les deux autres à leur exécution et à l'analyse :

- **QR1.1** : Comment concevoir un cas de test incluant des temps d'apprentissage et des temps d'évaluation, en intégrant l'interaction avec l'humain ?
- **QR1.2** : Comment concevoir un ensemble de cas de test suffisamment couvrant du champ d'application du programme apprenant tout en contrôlant la taille de cet ensemble ?
- **QR1.3** : Comment implanter un cas de test dans le but d'automatiser son exécution ?
- **QR2.1** : Comment automatiser l'exécution des tests en prenant en compte le non-déterminisme ?
- **QR2.2** : Comment mesurer la qualité des résultats obtenus ?

Notre contribution est centrée algorithmique au sens de [4] plutôt qu'humain : ce n'est pas l'utilisabilité et l'expérience utilisateur que nous cherchons à évaluer mais la qualité des modèles produits en fonction des données d'apprentissage. Nous ciblons dans cet article les questions **QR1.1**, **QR1.3**, **QR2.1** et **QR2.2**, à savoir la conception, l'implantation et l'exécution des cas de test. D'autres approches apportent des réponses à nos questions ; par exemple, le test métamorphique [8] s'intéresse à la vérification du comportement du modèle en l'absence d'oracle et cible la question **QR2.2**.

3.3 Test d'OCE, un programme apprenant en ligne avec l'utilisateur dans la boucle

Dans le cadre du projet de recherche OppoCompo, nous développons un « moteur de composition opportuniste » appelé OCE, qui apprend en ligne par renforcement à partir des retours de l'utilisateur. Dans le contexte des systèmes ambiants qui sont ouverts et fortement dynamiques par nature, par exemple une ville ou un bâtiment intelligent, OCE détecte les composants logiciels [25] qui peuplent l'environnement ambiant de l'utilisateur. Il les assemble automatiquement pour faire émerger une application de l'environnement. Comme les applications ne sont pas spécifiées ou demandées au préalable, il n'est pas possible de les connaître à l'avance. Pour prendre une décision, OCE s'appuie sur un modèle construit par apprentissage automatique lors des étapes qui précèdent l'étape de décision : lors d'une étape (appelée cycle), OCE propose une application à l'utilisateur humain qui peut l'accepter, la modifier ou la rejeter. OCE apprend de ce retour et met à jour le modèle afin d'améliorer les propositions des cycles suivants [30].

OCE est donc le programme apprenant qu'il nous faut tester. Pour cela, il faut évaluer les différents modèles qu'il construit et qu'il met à jour d'un cycle à un autre : ces modèles sont-ils capables de proposer des applications pertinentes pour l'utilisateur, c'est-à-dire conformes à ses préférences habituelles ? Comme un modèle ne se prête pas à la comparaison avec un modèle attendu ou à l'analyse de code, il doit être exécuté pour vérifier les applications qu'il fait émerger. Notons qu'il ne s'agit pas d'évaluer la qualité brute de ces applications (fonctionnalités, performance, sécurité, etc.), mais le fait qu'elles satisfont les attentes de l'utilisateur.

Grâce au test, nous cherchons à trouver et à corriger des défauts, à comparer différentes versions ou paramétrages, et finalement à instaurer de la confiance dans OCE. Mais, à travers le cas d'étude qu'est OCE, nous cherchons plus généralement à faire progresser l'état de l'art en matière de test de solutions apprenantes en ligne en interaction avec l'utilisateur humain.

Dans les sections suivantes, nous présentons une solution

pour concevoir, implanter et exécuter des cas de test.

4 Une approche basée sur les scénarios pour évaluer OCE

Cette section examine d'abord ce que sont les cas de test et leur structure. Elle présente ensuite deux outils, Maker et Runner, qui permettent de les implanter et de les exécuter automatiquement en interaction avec OCE.

4.1 Scénarios de test

Dans un contexte itératif tel que l'apprentissage en ligne, la conception d'un cas de test demande la définition d'une séquence d'interactions entre le programme apprenant et son environnement d'apprentissage (l'environnement ambiant et l'utilisateur dans notre cas). L'apprentissage en ligne nécessite un certain nombre d'interactions pour apprendre, donc pour dériver un modèle. Pour vérifier que le modèle se comporte comme attendu, d'autres interactions sont nécessaires. Nous appelons « scénario de test » cette séquence d'interactions dédiées à l'apprentissage et à l'évaluation [1]. Les interactions liées à l'apprentissage et à l'évaluation peuvent être imbriquées. Toutefois, dans cet article et à des fins de simplification, nous ne considérons qu'une phase d'apprentissage et une phase d'évaluation exécutées l'une après l'autre.

La **phase d'apprentissage** implique une séquence de cycles avec, pour chacun, une proposition d'OCE, un retour de l'utilisateur puis un apprentissage. Pour définir la phase d'apprentissage, chaque cycle doit être spécifié par : *(i)* la liste des composants logiciels peuplant l'environnement ambiant (variable d'un cycle à l'autre) avec leurs interfaces pour les assembler, *(ii)* pour éviter que le testeur n'ait à interagir en permanence avec OCE lors des tests, ce qui serait trop fastidieux et coûteux, l'« assemblage idéal » spécifie le résultat (la sortie) que l'utilisateur attendrait dans ce cas. À partir de l'assemblage des composants proposé par OCE et de l'assemblage idéal, OCE génère un retour et apprend, tout comme il le fait dans un fonctionnement normal. Au terme de cette phase, OCE a construit un modèle adapté aux configurations de l'environnement (y compris les retours de l'utilisateur). Le modèle est ensuite à évaluer. Pour simplifier les tests et l'analyse des résultats, la **phase d'évaluation** peut être réduite à un seul cycle, mais ce n'est pas obligatoire. Un cycle d'évaluation est défini par : *(i)* l'environnement ambiant comme dans la phase d'apprentissage et *(ii)* la sortie attendue appelée « assemblage attendu ». Ainsi, en donnant la sortie attendue, le concepteur du test se comporte comme un oracle. Des mesures de distance entre la sortie proposée par OCE et la sortie attendue sont calculées pour évaluer la pertinence (exactitude au sens de Zhang *et al.* [31]) de la décision d'OCE.

Cette approche à base de scénarios répond à la question de recherche **QR1.1**. De cette manière, il est possible de définir et de tester différents environnements ambiants et leur dynamique, ainsi que différents profils d'utilisateurs. Une fois implantés, ces scénarios sont destinés à être exécutés automatiquement. Il convient de noter que les problèmes liés à l'identification de scénarios de test représentatifs et à l'implication de l'utilisateur dans leur définition dépassent le cadre de cet article.

1. Dans le domaine du test de logiciels, le terme « scénario de test » désigne habituellement l'organisation et planification du processus de test dans le cadre d'un projet de développement. Notre définition du terme scénario de test est différente : il s'agit d'un scénario d'utilisation destiné à être testé, constituant ainsi un cas de test.

4.1.1 Exemple

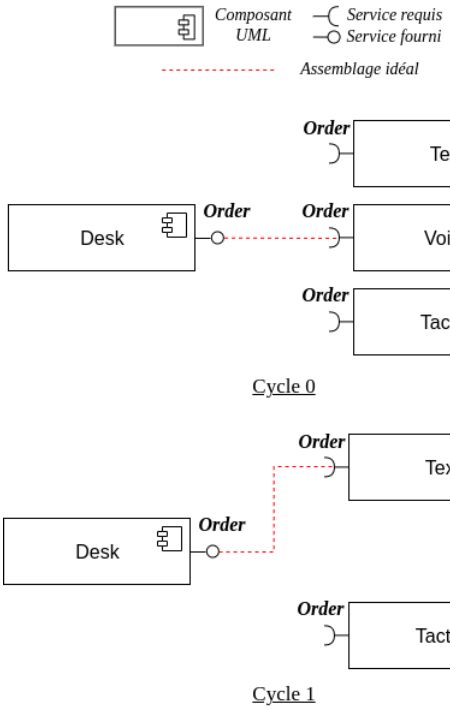

FIGURE 4 – Représentation UML [21] des cycles d'apprentissage du scénario exemple

Pour illustrer ce qu'est un scénario, prenons un exemple, tiré de [30] et réduit à des fins de simplification. Mary est au travail. Dans son environnement ambiant, il y a un composant logiciel [21] **Desk** qui fournit un service de réservation de salle appelé *Order* et trois composants **Text**, **Voice** et **Tactile** qui permettent de faire une demande de réservation (avec différents modes d'interaction pour l'utilisateur) et requièrent le service *Order*. Via le service *Order*, ces trois composants peuvent être assemblés avec **Desk**. Trois applications sont donc possibles (**Text-Desk**, **Voice-Desk**, **Tactile-Desk**) permettant à Mary de réserver une salle de réunion. En tant que testeurs, nous souhaitons par exemple vérifier que si Mary exprime une préférence pour **Voice-Desk**, alors, lorsqu'une situation similaire se présentera, OCE proposera à nouveau **Voice-Desk**.

Imaginons un scénario simple à trois cycles dans lequel **Voice** disparaît puis réapparaît. Les cycles 0 et 1 (figure 4) définissent la phase d'apprentissage. Le cycle 0 est défini par la liste des 4 composants et l'assemblage idéal **Voice-Desk** (l'expression de la préférence de Mary). Pour le cycle 1, les composants sont **Desk**, **Text** et **Tactile**. L'assemblage idéal est alors **Text-Desk**. Pour la phase d'évaluation, le

cycle 2 est défini par la liste composée de **Desk**, **Text** et **Voice**, et l'assemblage attendu est **Voice-Desk**. Dans ce qui suit, nous présentons deux outils qui supportent la définition et l'exécution d'un tel scénario.

4.2 Un outillage pour le test d'OCE

Les outils que nous avons développés pour tester le programme apprenant OCE sont présentés dans cette section. Le code source est disponible [2], ainsi qu'une courte vidéo [3] complétant cette section et démontrant leur utilisation dans le scénario présenté ci-dessus.

4.2.1 Maker

Il s'agit d'un outil interactif pour implanter des scénarios qui répond à la question **QR1.3**. Pour un cycle donné, le testeur peut réutiliser ou définir des composants fictifs (figure 5, panneau de gauche), les glisser (*drag and drop*) dans un cadre qui définit l'environnement et lier les services afin de définir l'assemblage idéal ou attendu (figure 5, panneau de droite). Des fonctionnalités telles que la possibilité de dupliquer un cycle réduisent la charge de travail du testeur. À partir d'une séquence de cycles, Maker génère un fichier JSON implantant le scénario.

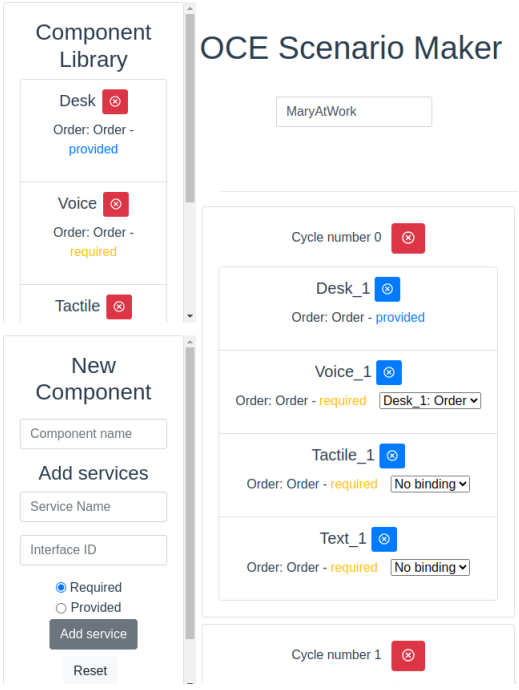

FIGURE 5 – Interface de Maker

4.2.2 Runner

Il s'agit d'une application Java qui, couplé avec OCE, permet l'exécution de scénarios au format JSON, comme ceux générés par Maker. Pour réduire l'impact de la nature non-déterministe de l'apprentissage automatique, l'exécution d'un scénario peut être répétée plusieurs fois et des valeurs moyennes des résultats mesurés sont calculées. Plusieurs paramètres doivent être définis, tels que les valeurs des paramètres d'apprentissage (par exemple, le taux d'exploration dans le cas de l'apprentissage par renforcement), la version d'OCE et le nombre de répétitions. Runner prend en paramètre une indication des cycles dédiés à l'apprentissage et des cycles dédiés à l'évaluation, cette indication définissant le moment des tests. Runner permet ainsi de répondre à la question **QR2.1**. Une fois les valeurs des paramètres définies, il fonctionne sans aucune autre intervention du testeur.

En réponse à la question **QR2.2**, pour évaluer la pertinence des applications proposées lors de l'exécution des cycles d'évaluation, Runner compare l'assemblage proposé par OCE et l'assemblage attendu en calculant un indice de similarité de Jaccard [4]. Il fournit un indice moyen sur l'ensemble des cycles d'évaluation d'un scénario. Cette mesure indique si les modèles construits ont fait des propositions pertinentes en fonction de ce qu'OCE a appris, c'est-à-dire en fonction des préférences de l'utilisateur dans différents environnements ambiants.

4.2.3 Architecture

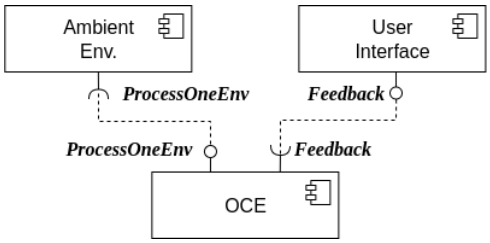

FIGURE 6 – Vue architecturale d'OCE en production

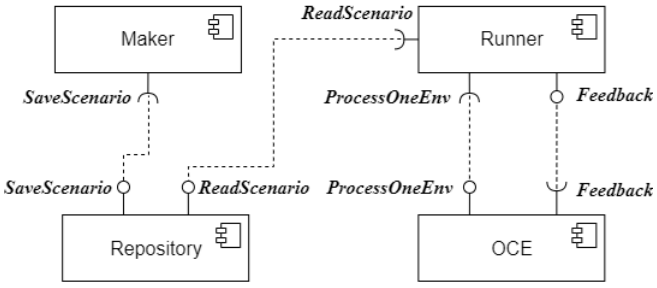

FIGURE 7 – Vue architecturale d'OCE en test

Les diagrammes de composants UML [21] (figures 6 et 7) montrent comment Maker et Runner ont été intégrés dans l'architecture du système logiciel OCE.

Dans la configuration de production (figure 6), lorsqu'il y a une variation de l'environnement ambiant, OCE reçoit du composant **Ambient Env.** la liste des composants présents dans l'environnement ambiant (service ***ProcessOneEnv***. L'assemblage construit par OCE est proposé à l'utilisateur qui fournit un retour sous la forme d'un assemblage (service ***Feedback***) via une interface graphique.

2. https://www.irit.fr/OppoCompo/resources/
3. https://www.irit.fr/OppoCompo/makerrunnerusecase2024/
4. https://en.wikipedia.org/wiki/Jaccard_index

L'architecture modulaire d'OCE permet de remplacer l'environnement ambiant et l'interface utilisateur par Runner pour effectuer les tests. Dans la configuration de test (figure 7), Runner exécute un scénario issu d'un répertoire de scénarios (service ***ReadScenario***). Le répertoire est alimenté par les scénarios implantés avec Maker (service ***SaveScenario***). Pour chaque cycle du scénario, Runner sollicite OCE pour obtenir une proposition d'assemblage (service ***ProcessOneEnv***). Suite à la proposition, qu'il évalue dans le cas des cycles d'évaluation, Runner fournit à OCE l'assemblage idéal ou attendu (service **Feedback**), que ce dernier traite comme un retour de l'utilisateur.

4.2.4 Expérimentation

Les outils Maker et Runner ont été utilisés lors d'une première campagne de test du comportement d'OCE. Nous avons défini une dizaine de scénarios visant à tester des cas de base, comme l'apprentissage d'un composant préféré, celui d'un composant à écarter ou la sensibilité à la nouveauté ; pour un même cas, nous avons fait varier la dynamique de l'environnement. Runner a été testé avec les scénarios créés avec Maker. Aucune défaillance de ces deux outils n'a été relevée. Ils ont simplifié le travail du testeur et accéléré la campagne en évitant de lancer manuellement OCE des centaines de fois, en permettant de déclencher des changements dans l'environnement et ainsi de simuler sa dynamique, et en fournissant une mesure de pertinence sans analyse laborieuse des connaissances d'OCE.
Ces tests ont permis de déceler certains défauts dans les prises de décision d'OCE. Ils ont également permis une évaluation initiale de l'apprentissage, d'affiner le paramétrage et de corriger des défauts dans le traitement du retour de l'utilisateur.

5 Travaux connexes

Cette section présente des travaux qui portent sur l'utilisation de scénarios pour tester des logiciels basés sur l'apprentissage automatique, et positionne notre proposition dans ce contexte. Des outils dédiés au développement et à l'expérimentation de solutions à base d'apprentissage par renforcement sont ensuite brièvement introduits et analysés par rapport à la question du test et de l'usage de scénarios.

5.1 Des scénarios pour tester

5.1.1 Le concept de scénario

C. Kaner [6] propose une définition du concept de scénario : un scénario est une histoire d'une personne qui essaie de faire quelque chose avec le produit testé. Hussain et al. [13] proposent une autre définition proche de la précédente, mais qui ne fait pas directement référence au test : un scénario est une description informelle d'une utilisation spécifique d'un logiciel ou d'une partie d'un logiciel par un utilisateur. Ici, les scénarios sont définis à partir des cas d'utilisation (des besoins de l'utilisateur) et sont utilisés pour dériver des cas de test.
Un scénario permet de décrire le déroulement complet d'une utilisation, donc de tester le logiciel de bout en bout dans sa globalité (test de niveau « système »). Il est décrit sous la forme d'une séquence d'interactions entre le logiciel et un utilisateur. C'est également le cas des scénarios de test d'OCE (cf. section 4), qui décrivent une séquence d'environnements ambiants (un environnement étant représenté par une liste de composants logiciels) mais aussi les assemblages idéaux qui eux modélisent les interactions entre un utilisateur et OCE.

5.1.2 Le test de modèles pour les véhicules autonomes

Nous retrouvons la notion de scénario dans le cadre de l'évaluation de comportements de véhicules autonomes basés sur l'apprentissage automatique [20]. Ulbrich *et al.* [28] décrivent un scénario comme un déroulement temporel d'une séquence de scènes, où chaque scène représente une configuration de l'environnement physique dans lequel un ou des véhicules autonomes doivent opérer. La description d'une scène comprend la position et la dynamique des autres entités présentes telles que des véhicules, des piétons, l'infrastructure routière et même les conditions météorologiques. La description d'un scénario comprend aussi les transitions entre les différentes scènes décrites par des actions ou des événements dans les scénarios, ainsi que les critères d'évaluation. Par exemple, dans un contexte de simulation, un scénario de test peut permettre de vérifier qu'un véhicule autonome peut circuler sans encombre d'un point A à un point B en présence d'autres véhicules, de piétons traversant la route et dans des conditions météorologiques défavorables.
Dans ce cadre, la notion de scénario ne correspond plus à une séquence d'interactions entre un utilisateur et un logiciel, mais plutôt à une séquence d'environnements. On peut noter que, dans le cadre de véhicules autonomes, les environnements sont hétérogènes et plus délicats à modéliser. D'autre part, dans nos scénarios de test, les actions et les événements entre les différents cycles ne sont pas explicitement spécifiés : les transitions entre les cycles sont représentées par des variations dans l'environnement ambiant, telles que l'apparition, la disparition ou la réapparition de composants, mais ces transitions restent implicites dans la description des scénarios. En revanche, les objectifs sont aussi décrits dans un scénario de test d'OCE grâce aux assemblages attendus dans les cycles d'évaluation, qui permettent de mesurer la pertinence des propositions d'OCE.

5.1.3 Analyse

Le principal avantage de l'utilisation de scénarios pour le test réside dans la capacité à évaluer des logiciels de bout en bout dans des situations particulières. Cette approche se révèle particulièrement adaptée pour évaluer des logiciels à base d'apprentissage automatique, en raison de leur nature « boîte noire ».
Cependant, nos objectifs de test à base de scénarios diffèrent de ceux des approches décrites dans la section 5.1.2. Alors que les scénarios de test pour les véhicules autonomes visent à évaluer les décisions prises par les véhicules, ce qui revient à tester les modèles construits antérieurement par apprentissage automatique, les scénarios de test d'OCE sont conçus pour évaluer le programme apprenant qu'est OCE, en testant sa capacité à produire de « bons »

modèles. Nos scénarios intègrent ainsi non seulement des phases d'évaluation mais également des phases (préalables) d'apprentissage.

5.2 Quelques outils

La littérature propose des outils pour le développement de solutions d'apprentissage par renforcement ; nous en avons sélectionné trois et nous étudions si et comment ils considèrent la question du test.

Gymnasium [5] est une librairie Python qui propose des environnements d'apprentissage par renforcement standardisés. L'évaluation d'une solution d'apprentissage se fait en la comparant à d'autres solutions, en les exécutant dans un ou plusieurs environnements proposés. Gymnasium ne propose aucun moyen pour réaliser cette comparaison, qui reste à la charge de l'utilisateur.

DotRL [22] permet l'évaluation d'algorithmes d'apprentissage standards (SARSA, QLearning…) ou personnalisés dans des environnements prédéfinis. Pour cela, le testeur choisit l'algorithme, l'environnement et ce qu'il veut mesurer, par exemple la récompense moyenne. Ce mode de fonctionnement se rapproche ainsi de Runner. En revanche aucun travail d'analyse n'est effectué à partir des valeurs mesurées, qui sont affichées sous leur forme brute. De plus, lors de l'exécution, il n'y a pas de distinction entre phase d'apprentissage et phase d'évaluation.

Cogment [1] prend en compte l'humain dans la boucle. Cette plate-forme facilite le développement de solutions dans lesquelles l'humain peut intervenir pour accélérer l'apprentissage. Cogment cible l'apprentissage interactif par renforcement mais son objectif n'est pour le moment que la mise en œuvre de solutions et il n'y a pas d'outil spécifique au test.

Les outils cités ci-dessus sont destinés au développement et à l'expérimentation de solutions d'apprentissage par renforcement. Ils ne permettent que de comparer des algorithmes dans des environnements prédéfinis. Seul DotRL propose des moyens d'évaluation, en restituant les valeurs mesurées pendant l'exécution de l'expérimentation.

La notion de scénario de test est absente dans ces outils. De plus, les environnements proposés sont pour la plupart stationnaires, alors qu'OCE opère dans des environnements qui sont, par nature, dynamiques et ouverts. Les scénarios de test d'OCE permettent d'intégrer ces caractéristiques, en donnant la possibilité au testeur de modéliser une séquence d'environnements avec sa dynamique (apparition d'un composant, disparition d'un composant, réapparition d'un composant…).

6 Bilan et perspectives

Dans ce papier, nous avons d'abord analysé la problématique du test dans le contexte de l'apprentissage automatique, en mettant particulièrement l'accent sur l'apprentissage en ligne et interactif. Nous nous sommes concentrés sur le test de programmes apprenants (par opposition au test de modèles).

Pour répondre à nos questions de recherche concernant la conception, l'implantation et l'exécution des tests d'un programme apprenant en interaction avec l'humain, nous avons proposé une approche à base de scénarios composés de phases d'apprentissage et de phases d'évaluation. Pour mettre en pratique cette approche et tester OCE, nous avons proposé deux outils : Maker et Runner. Ceux-ci sont opérationnels mais en cours d'évolution : plusieurs fonctionnalités sont actuellement développées ou à développer, telles que donner aux testeurs la possibilité de définir leurs propres mesures de qualité (par exemple précision, rappel…), ou de suivre l'exécution des scénarios cycle par cycle. Cette solution à base de scénarios a été conçue pour répondre spécifiquement au besoin d'évaluation d'OCE ; néanmoins, les principes que nous proposons pourraient s'appliquer au test d'autres solutions d'apprentissage interactif en ligne.

Notre proposition laisse encore un certain nombre de questions ouvertes. À ce stade, notre solution ne propose pas d'accompagnement pour la conception des scénarios : le testeur définit « à la main » tous les scénarios de test avec leurs objectifs, et en fixe le nombre. L'exécution d'un test est automatisée mais la conduite d'une campagne de test ne l'est pas. On peut ajouter que, dans le contexte dynamique et ouvert de l'apprentissage en ligne, le programme apprenant peut rencontrer de multiples situations qu'il peut être difficile d'anticiper, ou encore, pour ce qui concerne OCE, des environnements contenant un grand nombre de composants et de services. Un problème est donc de couvrir au mieux le champ d'application du programme apprenant et de tester des cas que le testeur ou l'utilisateur n'auraient peut-être pas envisagés. Pour cela, nous travaillons sur la génération et la sélection automatique de scénarios de test basées sur des modèles d'environnement avec leur dynamique.

Notre approche soulève également des questions relatives à l'implication de l'utilisateur dans le processus de test. Dans l'état actuel, les testeurs ont besoin non seulement de compétences en matière de test de logiciels et de compétences en apprentissage automatique, mais aussi de connaissances sur le domaine métier. Les utilisateurs finaux pourraient contribuer à la fois à la conception de scénarios et à l'interprétation des résultats. Une piste consisterait à permettre aux utilisateurs finaux de définir des scénarios dans un langage facile d'utilisation, qui seraient automatiquement traduits au format JSON à l'aide de techniques inspirées de l'ingénierie dirigée par les modèles.

Références

[1] AI Redefined, S. K. Gottipati, S. Kurandwad, C. Mars, G. Szriftgiser, and F. Chabot. Cogment : Open Source Framework For Distributed Multi-actor Training, Deployment & Operations. *CoRR*, abs/2106.11345, 2021.

[2] S. Amershi, M. Cakmak, W. B. Knox, and T. Kulesza. Power to the people : The role of humans in interactive machine learning. *Ai Magazine*, 35(4) :105–120, 2014.

[3] P. Ammann and J. Offutt. *Introduction to software testing*. Cambridge University Press, 2016.

[4] N. Boukhelifa, A. Bezerianos, and E. Lutton. Evaluation of Interactive Machine Learning Systems. In *Human and Machine Learning : Visible, Explainable, Trustworthy and Transparent*, pages 341–360. Springer, 2018.

[5] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. OpenAI Gym. *CoRR*, 2016.

[6] JD Cem Kaner. An introduction to scenario testing. *Florida Institute of Technology, Melbourne*, pages 1–13, 2013.

[7] J. Chen, J. Patra, M. Pradel, Y. Xiong, H. Zhang, D. Hao, and L. Zhang. A survey of compiler testing. *ACM Comput. Surv.*, 53(1), 2020.

[8] T. Y. Chen, S. C. Cheung, and S. M. Yiu. Metamorphic testing : a new approach for generating next test cases. *arXiv preprint arXiv :2002.12543*, 2020.

[9] J. A. Fails and D. R. Olsen Jr. Interactive machine learning. In *Proceedings of the 8th Int. Conf. on Intelligent User Interfaces*, pages 39–45, 2003.

[10] G. Giray. A software engineering perspective on engineering machine learning systems : State of the art and challenges. *J. of Systems and Software*, 180 :111031, 2021.

[11] D. Graham, R. Black, and E. van Veenendaal. *Foundations of Software Testing : ISTQB Certification*. Cengage, 4 edition, 2020.

[12] D. M. Hawkins. The problem of overfitting. *Journal of chemical information and computer sciences*, 44(1) :1–12, 2004.

[13] A. Hussain, A. Nadeem, and M. T. Ikram. Review on formalizing use cases and scenarios : Scenario based testing. In *2015 Int. Conf. on Emerging Technologies (ICET)*, pages 1–6. IEEE, 2015.

[14] F. Khomh, B. Adams, J. Cheng, M. Fokaefs, and G. Antoniol. Software engineering for machine-learning applications : The road ahead. *IEEE Software*, 35(5) :81–84, 2018.

[15] D. Marijan, A. Gotlieb, and M. K. Ahuja. Challenges of Testing Machine Learning Based Systems. In *Proc. of the 1st IEEE Artificial Intelligence Testing Conf.*, San Francisco, CA, USA, 2019. IEEE.

[16] S. Martínez-Fernández, J. Bogner, X. Franch, M. Oriol, J. Siebert, A. Trendowicz, A.-M. Vollmer, and S. Wagner. Software engineering for AI-based systems : a survey. *ACM Trans. on Software Engineering and Methodology (TOSEM)*, 31(2) :1–59, 2022.

[17] T. Mitchell. *Machine Learning*. McGraw-Hill, New York, 1997.

[18] C. Murphy, G. E. Kaiser, and M. Arias. An approach to software testing of machine learning applications. In *Int. Conf. on Software Engineering and Knowledge Engineering*, 2007.

[19] S. Nakajima. Generalized Oracle for Testing Machine Learning Computer Programs. In *Software Engineering and Formal Methods - SEFM 2017*, volume 10729 of *LNCS*, pages 174–179. Springer, 2017.

[20] D. Nalic, T. Mihalj, M. Bäumler, M. Lehmann, A. Eichberger, and S. Bernsteiner. Scenario based testing of automated driving systems : A literature survey. In *FISITA web Congress*, volume 10, 2020.

[21] OMG. *Unified Modeling Language*, chapter 11.6. 2017. `https://www.omg.org/spec/UML/2.5.1/PDF`.

[22] B. Papis and P. Wawrzyński. dotRL : A platform for rapid Reinforcement Learning methods development and validation. In *2013 Fed. Conf. on Computer Science and Information Systems (FEDCSIS)*, pages 129–136. IEEE, 2013.

[23] V. Riccio, G. Jahangirova, A. Stocco, N. Humbatova, M. Weiss, and P. Tonella. Testing machine learning based systems : a systematic mapping. *Empirical Software Engineering*, 25 :5193–5254, 2020.

[24] S. J. Russell and P. Norvig. *Artificial intelligence : A Modern Approach*. Pearson Education, Inc., 2010.

[25] I. Sommerville. Component-based software engineering. In *Software Engineering*, pages 464–489. Pearson Education, 10^{th} edition, 2016.

[26] K. Sugali. Software testing : Issues and challenges of artificial intelligence & machine learning. *Int. J. of Artificial Intelligence & Applications*, 12(1) :101–112, 2021.

[27] R. Sutton and A. Barto. *Reinforcement Learning : An Introduction*. MIT Press, 2nd edition, 2018.

[28] S. Ulbrich, T. Menzel, A. Reschka, F. Schuldt, and M. Maurer. Defining and substantiating the terms scene, situation, and scenario for automated driving. In *IEEE 18th Int. Conf. on intelligent transportation systems*, pages 982–988. IEEE, 2015.

[29] E. J. Weyuker. On testing non-testable programs. *The Computer Journal*, 25(4) :465–470, 1982.

[30] W. Younes, S. Trouilhet, F. Adreit, and J.-P. Arcangeli. Agent-mediated application emergence through reinforcement learning from user feedback. In *29th IEEE Int. Conf. on Enabling Technologies : Infrastructure for Collaborative Enterprises (WETICE)*, pages 3–8. IEEE Press, 2020.

[31] J. M. Zhang, M. Harman, L. Ma, and Y. Liu. Machine learning testing : Survey, landscapes and horizons. *IEEE Trans. on Software Engineering*, 48(1) :1–36, 2020.