Une enquête sur les techniques de vérification et de validation formelles pour l'Internet des objets

Moez Krichen

Une enquête sur les techniques de vérification et de validation formelles pour l'Internet des objets

Moez Krichen

Laboratoire ReDCAD, Université de Sfax, Tunisie
moez.krichen@redcad.org

Résumé. L'Internet des objets (IoT) a inauguré une nouvelle ère de dispositifs et de systèmes connectés, avec des applications allant de la santé aux transports. Cependant, la fiabilité et la sécurité de ces systèmes sont des préoccupations critiques qui doivent être abordées pour assurer leur fonctionnement sûr et efficace. Cet article présente une enquête sur les techniques de vérification et de validation formelles pour les systèmes IoT, en mettant l'accent sur les défis et les problèmes ouverts dans ce domaine. Nous donnons un aperçu des méthodes formelles et des techniques de test pour l'IoT et discutons du problème d'explosion d'états et des techniques pour y remédier. Nous examinons également l'utilisation de l'IA dans les tests logiciels et décrivons des exemples d'outils qui utilisent l'IA dans ce contexte. Enfin, nous abordons les défis et les problèmes ouverts en matière de vérification et de validation formelles pour l'IoT et présentons des orientations possibles pour la recherche future. Cet article d'enquête vise à fournir une compréhension complète de l'état actuel des techniques de vérification et de validation formelles pour les systèmes IoT et à mettre en évidence les domaines de recherche etde développement futurs.

1 Introduction

L'Internet des objets (IoT) est devenu une partie indispensable de la vie moderne, connectant des milliards de dispositifs à Internet et permettant une connectivité sans faille, une automatisation et une surveillance en temps réel (40; 37; 24; 25). L'IoT a transformé diverses industries, notamment la santé, les transports, les maisons intelligentes et l'automatisation industrielle, entraînant d'importantes améliorations en termes d'efficacité, de productivité et de confort (4). Cependant, les faiblesses des systèmes IoT, telles que les ressources limitées, les dispositifs hétérogènes et le manque de normalisation, les rendent sujets à des pannes dangereuses et des attaques (45).

Ces vulnérabilités ont entraîné d'importantes pertes et dommages dans le passé. Par exemple, en 2016, l'attaque de botnet Mirai a exploité les faiblesses de sécurité des dispositifs IoT pour lancer une attaque massive de déni de service distribué (DDoS), qui a perturbé Internet à travers le monde. L'attaque a causé une perte estimée à323 000 dollars par heure pour les entreprises touchées, totalisant plus de 100 millions de dollars de dommages (6). Un autre exemple est le ver Stuxnet, qui a ciblé les systèmes

de contrôle industriels et a causé des dommages physiques aux centrifugeuses nucléaires en Iran. L'attaque aurait retardé le programme nucléaire iranien de deux ans et causé plus d'un milliard de dollars de dommages (7).

Pour assurer la fiabilité, la sécurité et la sûreté des systèmes IoT, il est crucial d'appliquer des techniques de vérification et de validation formelles. Les méthodes formelles utilisent des techniques mathématiques pour modéliser et analyser rigoureusement les systèmes (64; 19; 18). Elles permettent aux concepteurs de système de spécifier le comportement et les propriétés du système en utilisant des notations mathématiques précises, telles que des formules logiques et des machines d'états. Les méthodes formelles peuvent ensuite utiliser des outils automatisés pour analyser le comportement et les propriétés du système, comme vérifierla cohérence, l'exhaustivité et la correction (22; 59). Elles peuvent également identifier les erreurs potentielles, les vulnérabilités et les attaques en explorant le comportement du système dans différents scénarios (3; 27). Les méthodes formelles peuvent aider à détecter les erreurs de conception tôt dans le processus de développement et à assurer que le système répond aux exigences et aux normes spécifiées (23).

Les tests sont une étape essentielle du processus de vérification et de validation, et différentes techniques ont été proposées pour tester efficacement les systèmes IoT (58; 26; 51). Par exemple, les techniques de test basées sur des modèles génèrent automatiquement des cas de test à partir d'un modèle formel du système, ce qui peut aider à obtenir une meilleure couverture de test et à réduire l'effort de test (16; 2). Les techniques de test de fuzz génèrent des données d'entrée aléatoires pour tester le système sous pression et identifier les vulnérabilitéset les cas limites (54; 14). De plus, les techniques de test en boucle fermée matériel-logiciel peuvent tester l'interaction entre le logiciel et les composants matériels des systèmes IoT, ce qui peut aider à détecter les problèmes d'intégration et les problèmes de compatibilité (52; 29; 65).

L'objectif de cet article est de fournir une enquête complète des techniques de vérification, de validation et de test formelles pour les systèmes IoT. Nous passerons en revue les méthodes et outils de pointe pour la modélisation, la spécification, la vérification et le test des systèmes IoT. Nous discuterons également de leurs forces et de leurs limites, et identifierons les défis ouverts et les orientations de recherche future dans ce domaine. En fournissant une vue d'ensemble de l'ensemble des méthodes et des outils de vérification, de validation et de test formelles pour l'IoT, cet article vise à aider les chercheurs et les praticiens à développer des systèmes IoT plus sûrs, plus fiables et plus dignes de confiance.

Les principales contributions de cet article sont résumées ci-dessous :

— Aperçu des techniques de vérification et de validation formelles pour les systèmes IoT
— Discussion des défis et des problèmes ouverts dans la vérification et la validation formelles pour les systèmes IoT
— Examen des méthodes formelles et des techniques de test pour les systèmes IoT
— Exploration de l'utilisation de l'IA dans les tests logiciels pour les systèmes IoT
— Identification des domaines de recherche et de développement futurs dans la vérification et la validation formelles pour les systèmes IoT

Le reste de l'article est structuré comme suit. Dans la Section 2, nous discutons des

éléments préliminaires liés à l'IoT, notamment la définition de l'IoT, ses caractéristiques et les défis qu'il présente. La Section 3 donne un aperçu des méthodes formelles pour l'IoT. Dans la Section IV, nous explorons les techniques de test pour l'IoT, y compris l'importance des tests, les différents types de tests et les défis des tests dans l'IoT. La Section 5 examine l'utilisation de l'IA dans les tests logiciels pour les systèmes IoT, notamment ses avantages et des exemples d'outils. Dans la Section 6, nous discutons des défis et des problèmes ouverts dans la vérification et la validation formelles pour les systèmes IoT et présentons des orientations de recherche future possibles. Enfin, nous concluons l'article dans la Section 7, en résumant les principales contributions et leur importance et en mettant en évidence les domaines de travail futurs.

2 Préliminaires liés à l'IoT

L'Internet des objets (IoT) a considérablement évolué au cours des dernières décennies, passant de la simple communication machine à machine (M2M) à un vaste réseau de dispositifs et de systèmes interconnectés. L'évolution de l'IoT a été stimulée par les avancées dans les technologies de communication, telles que les réseaux sans fil, les capteurs et l'informatique en nuage, ainsi que par la demande croissante d'automatisation et de surveillance en temps réel dans diverses industries.

L'une des principales caractéristiques de l'IoT est l'hétérogénéité, où des dispositifs de différents fabricants, avec des capacités et des ressources variables, doivent fonctionner ensemble de manière transparente. Cette hétérogénéité s'applique également aux données générées par ces dispositifs, qui peuvent avoir des formats, des structures et des sémantiques différents. Par conséquent, les systèmes IoT doivent utiliser des protocoles et des formats normalisés pour assurer l'interopérabilité et la compatibilité entre les dispositifs et les réseaux. Une autre caractéristique de l'IoT est la scalabilité, où les systèmes IoT doivent être capables de gérer un grand nombre de dispositifs et de données sans compromettre les performances et la fiabilité. Cette scalabilité peut être obtenue grâce à des architectures distribuées qui peuvent être mises à l'échelle horizontalement ou verticalement, selon les besoins de l'application. Cependant, la mise à l'échelle des systèmes IoT peut également augmenter leur complexité et introduire de nouveaux défis liés à la gestion des données, à la sécurité et à la confidentialité. Les systèmes IoT doivent également être résilients aux pannes et aux attaques, en raison de la nature critique de nombreuses applications, telles que les soins de santé et l'automatisation industrielle. Cette résilience peut être obtenue grâce à des mécanismes de redondance, de tolérance aux pannes et de reprise après sinistre qui peuvent garantir la disponibilité et l'intégrité des données et des services fournis par les systèmes IoT. Cependant, assurer la résilience peut également augmenter les coûts et la complexité des systèmes IoT, nécessitant des compétences et des outils spécialisés. Enfin, les systèmes IoT doivent être capables de fonctionner dans des environnements dynamiques et imprévisibles, tels que des environnements extérieurs ou des véhicules en mouvement. Cela nécessite que les systèmes IoT s'adaptent aux conditions changeantes, telles que les changements de topologie de réseau, les interférences ou la mobilité. De plus, les systèmes IoT doivent être capables de traiter les données en temps réel, ce qui nécessite des algorithmes efficaces de traitement et d'analyse des données.

L'architecture de l'IoT se compose généralement de quatre couches : la couche de perception, la couche de réseau, la couche intermédiaire et la couche applicative. La couche de perception comprend les capteurs et les actionneurs qui collectent et manipulent les données du monde physique. Ces capteurs peuvent être de différents types, tels que les capteurs de température, de pression, de lumière et de mouvement, et peuvent être connectés au réseau en utilisant différents protocoles de communication, tels que Zigbee, Wi-Fi et Bluetooth. La couche de réseau comprend les protocoles et les réseaux de communication qui permettent aux dispositifs de se connecter et de communiquer entre eux. Cette couche peut inclure différents types de réseaux, tels que les réseaux cellulaires, les réseaux satellitaires et les réseaux ad hoc, en fonctiondes exigences de l'application. La couche intermédiaire fournit les logiciels et les services nécessaires pour gérer les dispositifs et les données, tels que le stockage, le traitement et la sécurité des données. Cette couche peut inclure divers composants, tels que les courtiers de données, les files d'attente de messages et les outils d'analyse de données, qui peuvent faciliter l'intégration et l'analyse des données provenant de différentes sources. Enfin, la couche applicative comprend les logiciels et les services qui utilisent les données et les dispositifs pour fournir des services à valeur ajoutée, tels que les maisons intelligentes, la surveillance de la santé et l'automatisation industrielle. Cette couche peut inclure différents types d'applications, tels que les applications Web, les applications mobiles et les applications intégrées, en fonction de la plate-forme cible et des exigences de l'utilisateur.

Malgré les nombreux avantages de l'IoT, il existe également plusieurs limitations qui doivent être prises en compte. L'une des principales limitations est le manque de normalisation, qui peut entraver l'interopérabilité et la compatibilité entre différents systèmes IoT. L'absence d'un langage et d'une notation communs pour décrire les systèmes IoT peut également entraver l'adoption et l'intégration de méthodes formelles dans le processus de développement. Il est donc nécessaire de mener des efforts de normalisation pour établir un cadre commun pour la modélisation, la spécification, la vérification et les tests des systèmes IoT à l'aide de méthodes formelles. Une autre limitation concerne les risques de sécurité et de confidentialité associés à l'IoT, en raison de la grande surface d'attaque et des vulnérabilités de certains dispositifs et réseaux. Les systèmes IoT peuvent être sujets à divers types d'attaques, tels que les attaques par déni de service, les violations de données et les attaques de logiciels malveillants, qui peuvent compromettre la confidentialité, l'intégrité et la disponibilité des données et des services fournis par les systèmes IoT. De plus, les systèmes IoT peuvent collecter des données sensibles, telles que des informations de santé personnelles, des informations financières et des données de localisation, qui peuvent être mal utilisées si elles ne sont pas adéquatement protégées.

Les systèmes IoT sont également confrontés à des défis liés à la consommation d'énergie, car de nombreux dispositifs sont alimentés par batterie et doivent fonctionner pendant de longues périodes sans recharge. Cela nécessite que les systèmes IoT utilisent des techniques efficaces de gestion de l'énergie, telles que la modulation de puissance, les modes de veille et la récupération d'énergie, qui peuvent prolonger la durée de vie de la batterie des dispositifs et réduire leur impact environnemental. Enfin, la complexité des systèmes IoT peut rendre leur développement, leur test et leur main-

tenance difficiles, nécessitant des compétences et des outils spécialisés. Les systèmes IoT peuvent impliquer plusieurs couches, dispositifs, protocoles et normes, ce qui peut augmenter l'effort de test et rendre difficile une couverture de test exhaustive. De plus, les systèmes IoT peuvent être soumis à des changements de exigences, de technologies et de réglementations, qui peuvent nécessiter des mises à jour et une maintenance fréquentes.

3 Méthodes Formelles

Dans cette section, nous présentons un aperçu concis des formes les plus courantes de techniques formelles actuellement disponibles pour la communauté de recherche (64; 19).

- **Interprétation Abstraite (11) :** L'interprétation abstraite est une méthode formelle utilisée pour analyser et vérifier le comportement des programmes informatiques. C'est une technique qui consiste à approximer le comportement d'un programme en abstrayant certains de ses détails. Le but de l'interprétation abstraite est de prouver qu'un programme satisfait certaines propriétés, telles que la sécurité, la vivacité ou la terminaison. L'interprétation abstraite fonctionne en définissant un ensemble de valeurs abstraites qui représentent les états possibles d'un programme. Ces valeurs abstraites sont définies de manière à sur-approximer l'ensemble des valeurs concrètes possibles. Cela permet à l'interprétation abstraite de raisonner sur le comportement d'un programme sans l'exécuter réellement. L'un des principaux avantages del'interprétation abstraite est qu'elle peut être utilisée pour analyser des programmes trop complexes pour être analysés à l'aide d'autres méthodes. C'est parce que l'interprétation abstraite peut raisonner sur le comportement d'un programme à un niveau d'abstraction plus élevé, ce qui permet de gérer un espace d'état beaucoup plus grand.
- **Analyse Sémantique Statique (20) :** L'analyse sémantique statique est une méthode formelle utilisée pour analyser le comportement des programmes informatiques en examinant leur code source. Le but de l'analyse sémantique statique est de détecter les erreurs et les problèmes potentiels dans un programme avant qu'il ne soit exécuté. L'analyse sémantique statique fonctionne en analysant la syntaxe et la structure d'un programme pour en déduire sa signification. Cela se fait en construisant un modèle mathématique du comportement du programme, qui peut ensuite être utilisé pour raisonner sur ses propriétés. L'un des avantages de l'analyse sémantique statique est qu'elle peut être appliquée tôt dans le processus de développement, ce qui peut économiser du temps et des ressources. Endétectant les erreurs avant l'exécution d'un programme, l'analyse sémantique statique peut contribuer à garantir que le produit final est correct et fiable. Cependant, l'analyse sémantique statique peut être difficile car elle repose sur la capacité à raisonner sur des modèles mathématiques complexes du comportement du programme. Cela nécessite des connaissances et une expertise spécialisées en méthodes formelles et en analyse mathématique. De plus, l'exactitude de l'analyse sémantique statique dépend de la qualité du modèle utilisé, ce qui peut être difficile à construire pour des programmes complexes.

- **Vérification de Modèle (55) :** La vérification de modèle est une méthode formelle utilisée pour vérifier la correction d'un système en explorant de manière exhaustive ses comportements possibles. Elle fonctionne en construisant un modèle du système et en spécifiant les propriétés désirées que le système doit satisfaire. Le vérificateur de modèle explore ensuite systématiquement tous les états possibles du système pour déterminer si ces propriétés sont satisfaites pour tous les comportements possibles. La vérification de modèle est particulièrement utile pour vérifier des systèmes complexes, telsque des systèmes concurrents et distribués, où les méthodes de test traditionnelles peuvent ne pas être suffisantes. Elle peut également être utilisée pour vérifier les conceptions matérielles et les protocoles. L'un des principaux avantages de la vérification de modèle est qu'elle peut fournir une couverture complète de tous les comportements possibles, ce qui en fait un outil puissant pour garantir la correction de systèmes critiques.
- **Assistants de Preuve (17) :** Les assistants de preuve sont des outils logiciels qui aident les utilisateurs à construire et à vérifier des preuves mathématiques. Ils fournissent un langage formel pour exprimer des énoncés mathématiques et un ensemble de règles pour manipuler ces énoncés afin de construire des preuves. Les assistants de preuve sont utiles pour formaliser des théories mathématiques et vérifier leur correction. Ils peuvent également être utilisés pour vérifier la correction des conceptions de logiciels et de matériels. L'un des principaux avantages des assistants de preuve est qu'ils fournissent un niveau élevé d'assurance que la preuve est correcte, car la preuve est construite en utilisant des règles formelles et le logiciel vérifie la preuve pour en assurer la correction.
- **Vérification Déductive (56) :** La vérification déductive est une méthode formelle utilisée pour vérifier la correction d'un logiciel en construisant une preuve formelle que le logiciel satisfait ses spécifications. Elle fonctionne en commençant par les spécifications du logiciel et en construisant systématiquement une preuve que l'implémentation du logiciel satisfait ces spécifications. La vérification déductive est particulièrement utile pour garantir la correction de systèmes critiques de sécurité, tels que ceux utilisés dans l'aviation et les dispositifs médicaux. L'un des principaux avantages de la vérification déductive est qu'elle peut fournir un niveau élevé d'assurance que le logiciel est correct, car la preuve est construite en utilisant des règles formelles et la preuve peut être vérifiée par un ordinateur.
- **Conception par Raffinement (10; 9) :** La conception par raffinement est une méthode formelle utilisée pour développer des logiciels corrects en affinant itérativement une spécification abstraite du logiciel jusqu'à obtenir une implémentation détaillée. Elle fonctionne en commençant par une spécification de haut niveau du logiciel et en affinant cette spécification étape par étape jusqu'à obtenir une implémentation détaillée. La conception par raffinement peut contribuer à garantir que le logiciel répond à ses spécifications et est exempt d'erreurs. Elle peut également contribuer à garantir que le logiciel est maintenable et peut être facilement modifié en cas de changement des exigences. L'un des principaux avantages de la conception par raffinement est qu'elle fournit une approche systématique du développement de logiciels qui peut contribuer à

garantir que le produit final est correct et répond à ses spécifications.

— **Test Basé sur le Modèle (33; 35; 36; 34; 31; 38; 30; 39)** : Le test basé sur le modèle est une méthode formelle utilisée pour tester des logiciels en générant des cas de test à partir d'un modèle du logiciel. Elle fonctionne en construisant un modèle du logiciel puis en utilisant ce modèle pour générer automatiquement des cas de test qui testent différents aspects du logiciel. Le test basé sur le modèle peut contribuer à garantir que le logiciel répond à ses spécifications et est exempt d'erreurs. Il peut également contribuer à réduire le temps et les efforts nécessaires pour tester le logiciel. L'un des principaux avantages du test basé sur le modèle est qu'il fournit une approche systématique du test qui peut contribuer à garantir que le produit final est correct et répond à ses spécifications.

De plus, les méthodes formelles peuvent être classées en trois catégories : Complètes, Partielles et Asymptotiquement Complètes. Dans cette classification, les méthodes Complètes garantissent de fournir une réponse définitive à un problème donné, tandis que les méthodes Partielles peuvent ne pas fournir une réponse définitive mais peuvent fournir des informations utiles. Les méthodes Asymptotiquement Complètes ne sont pas garanties de trouver une solution, mais à mesure que la taille du problème augmente, la probabilité de trouver une solution approche 1. Voici quelques détails supplémentaires concernant ces trois classes :

— **Méthodes Complètes (12)** : Les méthodes Complètes sont des méthodes formelles qui garantissent de fournir une réponse définitive à un problème donné. Cela signifie que si un problème a une solution, une méthode complète la trouvera. Par exemple, la vérification de modèle est une méthode complète car elle peut explorer systématiquement tous les comportements possibles d'un système pour déterminer si une propriété donnée est vraie ou non. Différents types de vérification formelle complète existent, àsavoir : les méthodes basées sur SMT [1] (8), les méthodes basées sur MILP [2] (63), les méthodes basées sur l'optimisation, etc.

— **Méthodes Partielles (13)** : Les méthodes Partielles sont des méthodes formelles qui peuvent ne pas fournir une réponse définitive à un problème donné. Cela signifie qu'une méthode partielle peut ne pas être en mesure de déterminer si un problème a une solution ou non. Par exemple, l'interprétation abstraite est une méthode partielle car elle peut fournir une sur-approximation du comportement d'un programme, mais elle peut ne pas être en mesure de déterminer si le programme satisfait une propriété donnée ou non.

— **Méthodes Asymptotiquement Complètes** : Les méthodes Asymptotiquement Complètes sont des méthodes formelles qui ne garantissent pas de fournir une réponse définitive à un problème, mais à mesure que la taille du problème augmente, la probabilité de trouver une solution approche 1. Cela signifie que pour des problèmes très grands, une méthode asymptotiquement complète trouvera presque toujours une solution. Par exemple, la recherche heuristique est une méthode asymptotiquement complète car à mesure que la taille de l'espace de recherche augmente, la probabilité de trouver une solution approche 1, bien

1. SMT = Satisfiability Modulo Theory.
2. MILP = Mixed Integer Linear Programming.

qu'il n'y ait aucune garantie qu'une solution sera trouvée pour une instance de problème donnée.

Les avantages les plus importants de l'utilisation des approches formelles sont les suivants :

- **Abstraction :** Les approches formelles permettent l'abstraction, ce qui signifie qu'elles peuvent fournir une vue de haut niveau du système logiciel. Cela peut aider à gérer la complexité en cachant les détails non pertinents et en se concentrant sur les caractéristiques essentielles du système. L'abstraction facilite également la compréhension du comportement du système et l'identification d'erreurs et de défauts potentiels.
- **Analyse Rigoureuse :** Les méthodes formelles fournissent une approche rigoureuse et systématique pour analyser les systèmes logiciels. Cela signifie qu'elles utilisent des modèles et des techniques mathématiques bien définis pour analyser le logiciel, ce qui peut aider à garantir que l'analyse est précise et complète. L'analyse rigoureuse peut identifier des défauts et des erreurs qui peuvent être manqués par d'autres méthodes, telles que les tests ou les examens informels.
- **Découverte précoce de défauts :** Les méthodes formelles peuvent être appliquées tôt dans le processus de développement de logiciels, ce qui peut aider à identifier les défauts et les erreurs avant qu'ils ne deviennent plus difficiles et coûteux à corriger. La découverte précoce de défauts peut également contribuer à améliorer la qualité globale du logiciel et à réduire le risque de défauts qui pourraient entraîner des défaillances du système ou des risques pour la sécurité.
- **Garanties de correction :** Les méthodes formelles peuvent fournir des garanties de correction, ce qui signifie qu'elles peuvent prouver que le logiciel respecte ses spécifications et fonctionne correctement. Cela peut fournir un niveau élevé d'assurance que le logiciel est correct et fiable. Les garanties de correction sont particulièrement importantes pour les systèmes critiques pour la sécurité, où les erreurs ou les défauts pourraient avoir des conséquences graves.
- **Fiabilité :** Les méthodes formelles peuvent améliorer la fiabilité des systèmes logiciels en réduisant le risque d'erreurs et de défauts. Cela peut contribuer à garantir que le logiciel se comporte comme prévu et qu'il est robuste etrésistant aux entrées ou conditions inattendues. La fiabilité est particulièrement importante pour les systèmes qui doivent fonctionner en continu ou qui ne peuvent pas être facilement réparés ou remplacés.
- **Scénarios de test efficaces :** Les méthodes formelles peuvent aider à identifier les scénarios de test les plus efficaces pour un système logiciel. Cela peut réduire le temps et les efforts nécessaires pour tester le logiciel, tout en garantissant que le logiciel respecte ses spécifications et fonctionne correctement. Les scénarios de test efficaces peuvent également contribuer à améliorer la qualité globale du logiciel et à réduire le risque de défauts qui pourraient entraîner des défaillances du système ou des risques pour la sécurité. (31; 33; 32).
- **Maintenabilité :** Les méthodes formelles peuvent améliorer la maintenabilité des systèmes logiciels en fournissant une spécification claire et précise du comportement du système. Cela peut faciliter la modification ou la refonte du logiciel sans introduire d'erreurs ou de défauts. Les méthodes formelles peuvent égale-

ment contribuer à garantir que les modifications ne violent pas les spécifications ou les exigences du système.
— **Réutilisabilité :** Les méthodes formelles peuvent améliorer la réutilisabilité des composants logiciels en fournissant une spécification claire et précise de leur comportement. Cela signifie que les composants logiciels peuvent être réutilisés dans différents contextes sans introduire d'erreurs ou de défauts. Les méthodes formelles peuvent également contribuer à garantir que les composants réutilisés se comportent correctement dans tous les contextes.
— **Normalisation :** Les méthodes formelles peuvent fournir une approche normalisée pour le développement et la vérification de logiciels. Cela signifie que les systèmes logiciels peuvent être développés et vérifiés en utilisant un ensemble commun de techniques et d'outils, ce qui peut améliorer l'interopérabilité et réduire le risque d'erreurs ou de problèmes de compatibilité.
— **Confiance :** Les méthodes formelles peuvent fournir aux développeurs et aux parties prenantes une confiance dans la correction et la fiabilité du système logiciel. Cela peut accroître la confiance dans le système logiciel et réduire le risque de conséquences négatives, telles que des défaillances du système ou des risques pourla sécurité.

4 Techniques de test

Le test est une partie essentielle du processus de développement de logiciels, car il contribue à garantir que les systèmes logiciels sont fiables, performants et répondent aux besoins de leurs utilisateurs. Il existe de nombreuses approches différentes pour tester les logiciels, chacune ayant ses propres objectifs et procédures spécifiques (Figure **??**) :

— **Test unitaire (28) :** Le test unitaire est une méthode de test qui se concentre sur les unités ou les composants individuels d'un système logiciel. L'objectif du test unitaire est de garantir que chaque composant du système fonctionne comme prévu et répond à ses normes. Le test unitaire est généralement réalisé en créant et en exécutant des cas de test pour chaque unité. Le principal avantage du test unitaire est de détecter les erreurs et lesdéfauts tôt dans le processus de développement, ce qui les rend plus faciles et moins coûteux à rectifier. Le plus grand inconvénient du test unitaire est qu'il peut ne pas détecter les erreurs ou les défauts qui se produisent lorsque les unités sont combinées.
— **Test d'intégration (57) :** Le test d'intégration est une méthode de test qui se concentre sur les interactions de plusieurs unités ou composants d'un système logiciel. L'objectif du test d'intégration est de garantir que le système dans son ensemble fonctionne comme prévu et que les unités fonctionnent correctement lorsqu'elles sont combinées. Le test d'intégration est souvent réalisé en testant différentes combinaisons d'unités et en vérifiant qu'elles fonctionnent comme prévu. Le principal avantage du test d'intégration est qu'il peut détecter les erreurs ou les défauts qui se produisent lorsque les unités sont combinées, ce qui les rend plus faciles et moins coûteux à corriger.
— **Test d'acceptation (62) :** Le test d'acceptation est une méthode permettant de déterminer si un système logiciel répond à ses exigences et spécifications.

L'objectif du test d'acceptation est de garantir que le système est acceptable pour ses parties prenantes et répond à leurs besoins. Le test d'acceptation est souvent réalisé en soumettant le système à des tests dans un environnement réel et en vérifiant qu'il satisfait les exigences et spécifications. Le principal avantage du test d'acceptation est de s'assurer que le système répond aux besoins de ses parties prenantes.

— **Test fonctionnel (61) :** Le test fonctionnel est une approche de test qui se concentre sur la fonctionnalité d'un système logiciel. L'objectif du test fonctionnel est de garantir que le système fonctionne correctement et répond à ses exigences et spécifications. Le test fonctionnel est généralementréalisé en testant le système à l'aide d'un ensemble de cas de test prédéfinis qui couvrent tous les aspects de sa fonctionnalité. Le principal avantage du test fonctionnel est qu'il peut garantir que le système fonctionne correctement et répond à ses exigences et spécifications.

— **Test d'utilisabilité (21) :** Le test d'utilisabilité est une approche de test qui se concentre sur la facilité d'utilisation d'un système logiciel. L'objectif du test d'utilisabilité est de garantir que le système est facile à utiliser et répond aux besoins de ses utilisateurs. Le test d'utilisabilité est généralement réalisé en testant le système avec un groupe d'utilisateurs représentatifs et en observant leur interaction avec le système. Le principal avantage du test d'utilisabilité est qu'il peut garantir que le système est facile à utiliser et répond aux besoins de ses utilisateurs.

— **Test de stress (50; 47; 48; 46; 49) :** Le test de stress est une approche de test qui se concentre sur la façon dont un système logiciel se comporte sous stress ou sous une charge importante. L'objectif du test de stress est de garantir que le système peut gérer des volumes de trafic ou de demandes élevés sans planter ou échouer. Le test de stress est généralement réalisé en testant le système avec un volume élevé de trafic ou de demandes et en observant son comportement. Le principal avantage du test de stress est qu'il peut garantir que le système est fiable et peut gérer des volumes de trafic ou de demandes élevés. Le principal inconvénient du test de stress est qu'il peut ne pas détecter les erreurs ou les défauts qui se produisent dans des conditions de fonctionnement normales.

— **Test de performance (5) :** Le test de performance est une méthode permettant de déterminer le bon fonctionnement d'un système logiciel dans des conditions d'utilisation régulières. L'objectif du test de performance est de garantir que le système est réactif et répond aux besoins de ses utilisateurs. Typiquement, le test de performance est réalisé en soumettant le système à une charge représentative et en évaluant sa performance. Le principal avantage du test de performance est qu'il assure que le système est réactif et fonctionne correctement pour ses utilisateurs. Le principal inconvénient du test de performance est qu'il peut manquer des erreurs ou des problèmes qui se produisent lorsque le système est sous pression ou surchargé.

— **Test de régression (41; 44; 42; 43) :** Le test de régression est une méthode de test qui vise à déterminer si les modifications apportées à un système logiciel ont introduit de nouvelles erreurs ou des défauts. L'objectif du test de régression

est de garantir que le système continue de fonctionner correctement après les modifications qui y ont été apportées. Le test de régression est souvent réalisé en retestant le système avec un ensemble de cas de test spécifiés après les modifications qui yont été apportées. Le principal avantage du test de régression est qu'il assure que le système continue de fonctionner correctement après les modifications qui y ont été apportées. Le principal inconvénient du test de régression est qu'il peut ne pas détecter les erreurs ou les problèmes qui surviennent lorsque le système est sous pression ou sous contrainte.

5 Utilisation de l'IA dans les tests logiciels

L'utilisation de l'Intelligence Artificielle (IA) dans les tests logiciels gagne en popularité ces dernières années (15; 60; 1; 53). L'IA peut aider à automatiser diverses tâches liées aux tests logiciels, telles que la génération de cas de test, l'exécution de tests et l'analyse des résultats. L'un des principaux avantages de l'utilisation de l'IA dans les tests logiciels est la capacité à améliorer la couverture et la qualité des tests, car l'IA peut analyser de grandes quantités de données et identifier des modèles et des anomalies qui ne seraient pas évidents pour les testeurs humains (15). Cela peut permettre une meilleure détection des défauts et des vulnérabilités, réduisant ainsi le risque de défaillances logicielles et de temps d'arrêt.

Un autre avantage de l'utilisation de l'IA dans les tests logiciels est la capacité à réduire le temps et les efforts nécessaires pour les tests (60). L'IA peut automatiser des tâches répétitives et chronophages, telles que les tests de régression, permettant aux testeurs de se concentrer sur des tâches plus complexes et créatives, telles que les tests exploratoires. Cela peut conduire à des cycles de sortie plus rapides et à une réduction du temps de mise sur le marché, ce qui est crucial dans l'industrie du développement de logiciels d'aujourd'hui, qui est en constante évolution.

De plus, l'IA peut également aider à améliorer l'efficacité et l'efficience des équipes de test, car elle peut fournir des informations et des recommandations basées sur l'analyse de données et des algorithmes d'apprentissage automatique (15). Cela peut aider les testeurs à hiérarchiser leurs efforts de test et à se concentrer sur les zones les plus critiques et susceptibles de présenter des défauts. De plus, l'IA peut également aider à réduire les coûts de test, car elle peut identifier les défauts et les vulnérabilités tôt dans le cycle de développement, réduisant ainsi le besoin de travaux de révision et de maintenance coûteux.

5.1 Avantages

L'utilisation de l'Intelligence Artificielle (IA) dans les tests logiciels offre de nombreux avantages qui peuvent aider à améliorer l'efficacité, l'efficience et la qualité du développement de logiciels. Dans cette section, nous discuterons de quelques-uns des principaux avantages qui peuvent être obtenus en utilisant des stratégies d'IA pour les tests logiciels.

— **Génération automatique de cas de test :** La génération automatique de cas de test est l'un des avantages les plus significatifs de l'utilisation de l'IA dans

les tests logiciels. L'IA peut analyser le code et identifier les zones potentielles de faiblesse, ce qui lui permet de générer des cas de test pouvant tester minutieusement le logiciel. Cela peut faire économiser des quantités importantes de temps et d'efforts qui seraient autrement consacrés à la rédaction manuelle de cas de test. De plus, les cas de test générés par l'IA peuvent souvent couvrir plus de scénarios et de cas limites que les cas de test écrits par des humains, ce qui entraîne des tests plus complets et une meilleure qualité logicielle.

— **Réduction du temps de mise sur le marché :** L'utilisation de l'IA dans les tests logiciels peut aider à réduire le temps de mise sur le marché des produits logiciels. En automatisant des tâches répétitives et chronophages, telles que les tests de régression, l'IA peut aider à accélérer le processus de test. Cela peut aider les entreprises de développement de logiciels à sortir leurs produits plus rapidement, en gagnant un avantage dans le marché concurrentiel. De plus, une mise sur le marché plus rapide peut entraîner une augmentation des revenus et une amélioration de la satisfaction des clients, car les clients sont plus susceptibles de choisir des produits qui sont rapidement mis sur le marché et régulièrement mis à jour avec de nouvelles fonctionnalités.

— **Rétroaction/Feedback précoce :** Un autre avantage de l'utilisation de l'IA dans les tests logiciels est la capacité à fournir un retour d'information précoce sur la qualité du logiciel. L'IA peut détecter les défauts et les vulnérabilités tôt dans le cycle de développement, permettant aux développeurs de les corriger avant qu'ils ne deviennent des problèmes majeurs. Cela peut aider à améliorer la qualité et la fiabilitédu logiciel, entraînant une meilleure satisfaction des clients. De plus, la détection précoce des problèmes peut aider à réduire les coûts et les efforts nécessaires pour les corriger plus tard dans le processus de développement.

— **Analyse prédictive :** L'analyse prédictive est un autre avantage de l'utilisation de l'IA dans les tests logiciels. L'IA peut analyser les données historiques et en temps réel pour prédire le comportement futur d'un système logiciel. Cela peut aider à identifier les problèmes potentiels avant qu'ils ne surviennent, permettant aux développeurs de prendre des mesures préventives pour éviter les temps d'arrêt ou les pannes du système. De plus, l'analyse prédictive peut aider à optimiser les performances et l'efficacité des systèmes logiciels, entraînant de meilleures expériences utilisateur et une amélioration de la satisfaction des clients.

— **Plateforme intégrée :** L'utilisation de l'IA dans les tests logiciels peut aider à intégrer différents outils et plateformes de test. L'IA peut aider à unifier différentes méthodes de test, telles que les tests unitaires, les tests d'intégration et les tests système. Cela peut aider les entreprises de développement de logiciels à économiser du temps et à réduire les coûts en utilisant une plateforme de test unique et intégrée. De plus, une plateforme de test intégrée peut fournir une vue holistique de la qualité du logiciel, permettant aux développeurs d'identifier et de résoudre les problèmes plus efficacement.

— **Réduction des tests basés sur l'interface utilisateur (UI) :** L'IA peut aider à réduire la nécessité de tests basés sur l'interface utilisateur, qui sont souvent chronophages et coûteux. En automatisant les tests de back-end, l'IA peut

aider à identifier les problèmes sans avoir besoin de tests d'interface utilisateur étendus, réduisant ainsi l'effort de test global requis. De plus, la réduction de la nécessité de tests basés sur l'interface utilisateur peut aider à améliorer l'efficacité des équipes de test, leur permettant de se concentrer sur des tâches de test plus complexes et critiques.

- **Meilleure couverture de code :** L'IA peut aider à améliorer la couverture de code en identifiant les zones qui ne sont pas suffisamment couvertes par les cas de test existants.Cela peut aider à garantir que toutes les parties du logiciel sont testées minutieusement, réduisant ainsi le risque de problèmes et de vulnérabilités. De plus, une meilleure couverture de code peut conduire à une meilleure qualité et fiabilité du logiciel, améliorant l'expérience utilisateur globale et la satisfaction des clients.
- **Amélioration de la fiabilité :** En automatisant les tâches de test, l'IA peut aider à améliorer la fiabilité des produits logiciels. Les tests automatisés peuvent détecter les défauts et les vulnérabilités qui pourraient être ignorés par les tests manuels, conduisant à des produits logiciels plus fiables et stables. De plus, une amélioration de la fiabilité peut aider à réduire les coûts et les efforts nécessaires pour la maintenance et le support, améliorant ainsi l'efficacité globale et l'efficience des équipes de développement de logiciels.
- **Amélioration de la qualité :** L'utilisation de l'IA dans les tests logiciels peut aider à améliorer la qualité globale des produits logiciels. En détectant les défauts et les vulnérabilités tôt dans lecycle de développement, l'IA peut aider à garantir que les produits logiciels sont de haute qualité et répondent aux attentes du client. De plus, une amélioration de la qualité peut conduire à une meilleure satisfaction des clients, une augmentation des revenus et un avantage concurrentiel sur le marché.
- **Tests de validation visuelle automatisés :** L'IA peut également être utilisée pour les tests de validation visuelle automatisés, qui consistent à comparer la sortie visuelle d'un système logiciel aux résultats attendus. Cela peut aider à identifier les défauts visuels et les incohérences, améliorant ainsi la qualité globale et l'expérience utilisateur du logiciel. De plus, les tests de validation visuelle automatisés peuvent aider à réduire l'effort requis pour les tests visuels manuels, permettant aux équipes de test de se concentrer sur des tâches de test plus complexes et critiques.

Dans l'ensemble, l'utilisation de l'IA dans les tests logiciels offre de nombreux avantages, notamment une mise sur le marché plus rapide, une meilleure fiabilité et qualité, ainsi qu'une réduction des efforts et des coûts de test. À mesure que la technologie de l'IA continue d'évoluer, il est probable que de plus en plus d'avantages deviendront évidents, en faisant ainsi un outil de plus en plus précieux pour le développement et les tests logiciels.

5.2 Exemples d'outils

Il existe de nombreux outils de test basés sur l'IA disponibles sur le marché qui peuvent aider à améliorer l'efficacité et l'efficience des tests logiciels. Ces outils utilisent

l'IA pour automatiser diverses tâches de test, telles que la génération de cas de test, l'exécution de tests et la détection de défauts. Voici quelques exemples :

- **Applitools :** Applitools est un outil de test visuel alimenté par l'IA qui permet aux testeurs de détecter automatiquement les défauts visuels et les incohérences dans les applications web et mobiles. L'outil utilise des algorithmes de vision par ordinateur pour comparer des captures d'écran de l'application sur plusieurs appareils, navigateurs et résolutions, et identifier les différences qui peuvent indiquer un défaut. Applitools peut également s'intégrer à des frameworks de test populaires, tels que Selenium et Appium, permettant aux testeurs d'intégrer facilement les tests visuels dans leurs processus de test existants. En outre, Applitools fournit un tableau de bord qui met en évidence les problèmes visuels et permet aux testeurs de suivre et de gérer facilement les défauts. En utilisant Applitools, les testeurs peuvent améliorer leur couverture et leur précision de test visuel, conduisant finalement à de meilleurs produits logiciels et une plus grande satisfaction des clients.
- **Appvance IQ :** Appvance IQ est un outil de test basé sur l'IA qui permet aux testeurs de générer et d'exécuter automatiquement des cas de test sur plusieurs plateformes et environnements. L'outil utilise des algorithmes avancés d'apprentissage automatique pour analyser le comportement des utilisateurs et générer des cas de test qui couvrent les cas d'utilisation les plus critiques et les plus courants. Appvance IQ peut également détecter les défauts et les vulnérabilités, et fournir des informations et des recommandations pour améliorer la qualité logicielle. En outre, Appvance IQ fournit un tableau de bord qui permet aux testeurs de suivre et de gérer facilement les défauts, ainsi que de visualiser des rapports et des analyses détaillés sur leurs activités de test. En utilisant Appvance IQ, les testeurs peuvent gagner du temps et des efforts sur la création et la maintenance des cas de test, tout en améliorant leur couverture et leur précision de test.
- **Functionize :** Functionize est un outil de test basé sur l'IA qui permet aux testeurs de générer et d'exécuter des cas de test de manière autonome, ainsi que de détecter et de prioriser les défauts. L'outil utilise des algorithmes avancés d'apprentissage automatique pour analyser le comportement des utilisateurs et générer des cas de test qui couvrent les cas d'utilisation les plus critiques et les plus courants. Functionize peut également détecter les défauts et les vulnérabilités, et les prioriser automatiquement en fonction de leur gravité, réduisant ainsi les efforts nécessaires pour le tri manuel des défauts. En outre, Functionize fournit un tableau de bord qui permet aux testeurs de suivre et de gérer facilement les défauts, ainsi que de visualiser des rapports et des analyses détaillés sur leurs activités de test. En utilisant Functionize, les testeurs peuvent améliorer leur efficacité et leur efficience de test, tout en réduisant le temps et les efforts nécessaires pour la création et la maintenance des cas de test.
- **Mabl :** Mabl est un outil de test basé sur l'IA qui permet aux testeurs d'identifier et de prioriser automatiquement les problèmes, ainsi que de générer et de maintenir des cas de test. L'outil utilise des algorithmes avancés d'apprentissage automatique pour analyser le comportement des utilisateurs et générer des cas

de test qui couvrent les cas d'utilisation les plus critiques et les plus courants. Mabl peut également détecter automatiquement les problèmes et les vulnérabilités, et les prioriser en fonction de leur gravité, réduisant ainsi les efforts nécessaires pour le tri manuel des défauts. En outre, Mabl fournit un tableau de bord qui permet aux testeurs de suivre et de gérer facilement les défauts, ainsi que de visualiser des rapports et des analyses détaillés sur leurs activités de test. En utilisant Mabl, les testeurs peuvent améliorer leur efficacité et leur efficience de test, tout en réduisant le temps et les efforts nécessaires pour la création et la maintenance des cas de test.

— **ReTest :** ReTest est un outil de test basé sur l'IA qui permet aux testeurs d'analyser les exigences logicielles et de générer des cas de test qui couvrent toutes les combinaisons possibles de paramètres d'entrée. L'outil utilise des algorithmes avancés d'apprentissage automatique pour analyser les exigences et générer des cas de test qui couvrent toutes les combinaisons possibles de paramètres d'entrée, assurant ainsi une couverture de test complète. ReTest peut également détecter automatiquement les défauts et les vulnérabilités, et fournir des informations et des recommandations pour améliorer la qualité logicielle. En outre, ReTest fournit un tableau de bord qui permet aux testeurs de suivre et de gérer facilement les défauts, ainsi que de visualiser des rapports et des analyses détaillés sur leurs activités de test. En utilisant ReTest, les testeurs peuvent améliorer leur efficacité et leur efficience de test, tout en assurant une couverture de test complète et en réduisant le risque de défauts et de vulnérabilités.

— **Sauce Labs :** Sauce Labs est un outil de test basé sur l'IA qui fournit des tests automatisés pour les applications web et mobiles. L'outil utilise des algorithmes avancés d'apprentissage automatique pour générer et exécuter automatiquement des cas de test sur plusieurs plateformes et environnements, assurantune couverture de test complète. Sauce Labs peut également détecter les défauts et les vulnérabilités, et fournir des informations et des recommandations pour améliorer la qualité logicielle. En outre, Sauce Labs fournit un tableau de bord qui permet aux testeurs de suivre et de gérer facilement les défauts, ainsi que de visualiser des rapports et des analyses détaillés sur leurs activités de test. En utilisant Sauce Labs, les testeurs peuvent améliorer leur efficacité et leur efficience de test, tout en assurant une couverture de test complète sur plusieurs plateformes et environnements.

— **Test.AI :** Test.AI est un outil de test basé sur l'IA qui permet aux testeurs de générer et d'exécuter des cas de test, ainsi que de détecter et de prioriser les défauts. L'outil utilise des algorithmes avancés d'apprentissage automatique pour analyser le comportement de l'utilisateur et générer des cas de test qui couvrent les cas d'utilisation les plus critiques et les plus courants. Test.AI peut également détecter les défauts et les vulnérabilités, et les prioriser en fonction de leur gravité, réduisant ainsi l'effort nécessaire pour le tri manuel des défauts. De plus, Test.AI fournit un tableau de bord qui permet aux testeurs de suivre et de gérer facilement les défauts, ainsi que de consulter des rapports détaillés et des analyses de leurs activités de test. En utilisant Test.AI, les testeurs peuvent améliorer leur efficacité et leur efficience de test, tout en réduisant le temps et

l'effort nécessaires à la création et à la maintenance des cas de test.

— **Testim :** Testim est un outil de test basé sur l'IA qui permet aux testeurs de générer et de maintenir descas de test, réduisant ainsi le besoin de créer et de maintenir manuellement des cas de test. L'outil utilise des algorithmes avancés d'apprentissage automatique pour analyser le comportement de l'utilisateur et générer automatiquement des cas de test qui couvrent les cas d'utilisation les plus critiques et les plus courants. Testim peut également maintenir les cas de test à mesure que l'application évolue, en veillant à ce qu'ils restent à jour et pertinents. De plus, Testim fournit un tableau de bord qui permet aux testeurs de suivre et de gérer facilement les défauts, ainsi que de consulter des rapports détaillés et des analyses de leurs activités de test. En utilisant Testim, les testeurs peuvent économiser du temps et des efforts sur la création et la maintenance des cas de test, tout en améliorant leur efficacité et leur efficience de test.

— **Tricentis Tosca :** Tricentis Tosca est un outil de test basé sur l'IA qui permet aux testeurs d'automatiser les tests de bout en bout pour les applications web et mobiles. L'outil utilise des algorithmes avancés d'apprentissage automatique pour générer et exécuter automatiquement des cas de testsur plusieurs plates-formes et environnements, assurant une couverture de test complète. Tricentis Tosca peut également détecter les défauts et les vulnérabilités, et fournir des informations et des recommandations pour améliorer la qualité du logiciel. De plus, Tricentis Tosca fournit un tableau de bord qui permet aux testeurs de suivre et de gérer facilement les défauts, ainsi que de consulter des rapports détaillés et des analyses de leurs activités de test. En utilisant Tricentis Tosca, les testeurs peuvent améliorer leur efficacité et leur efficience de test, tout en assurant une couverture de test complète sur plusieurs plates-formes et environnements.

Ces outils démontrent les diverses façons dont l'IA peut être appliquée aux tests logiciels, notamment les tests visuels, la génération de cas de test, la détection de défauts et les tests automatisés sur plusieurs plates-formes et environnements. En utilisant des outils de test basés sur l'IA, les développeurs et les testeurs peuvent améliorer l'efficacité, l'efficience et la qualité des tests logiciels, ce qui conduit finalement à de meilleurs produits logiciels et à une plus grande satisfaction desclients.

6 Défis et enjeux ouverts

L'Internet des objets (IoT) présente un ensemble unique de défis pour les techniques de vérification et de validation formelles. L'un des principaux défis est la complexité et l'hétérogénéité des systèmes IoT. Les systèmes IoT peuvent impliquer de nombreux dispositifs et réseaux, chacun avec des matériels, logiciels et protocoles de communication différents. Cela rend difficile le développement d'un cadre unifié de vérification formelle qui peut être appliqué à tous les dispositifs. De plus, le manque de normalisation dans les dispositifs et les réseaux IoT rend difficile le développement de modèles formels qui capturent précisément le comportement de ces systèmes. Par exemple, différents dispositifs peuvent avoir des protocoles de communication différents ou utiliser des formats de données différents, ce qui rend difficile le développement d'un modèle formel unifié qui peut être appliqué à tous les dispositifs. Un autre défi est la nature

dynamique des systèmes IoT. Les dispositifs peuvent rejoindre ou quitter le système à tout moment, et le comportement du système peut changer en fonction du contexte et de l'environnement. Cela rend difficile le développement d'un modèle formel statique qui peut capturer précisément le comportement du système. Pour relever ces défis, les chercheurs ont développé des modèles formels spécifiques aux dispositifs et des techniques de vérification qui peuvent être adaptées aux caractéristiques spécifiques de chaque dispositif. De plus, ils ont développé des interfaces et des protocoles normalisés qui permettent l'interopérabilité entre les dispositifs et les réseaux, améliorant ainsi la précision et la fiabilité des modèles formels.

Un autre défi dans les techniques de vérification et de validation formelles pour l'IoT est le problème d'explosion d'état. Les systèmes IoT peuvent impliquer un grand nombre de dispositifs et d'états, rendant difficile leur analyse exhaustive. Ce problème peut être résolu à l'aide de techniques d'abstraction, de modularisation et de détection de symétrie. L'abstraction implique de simplifier le modèle du système en supprimant les fonctionnalités superflues, tandis que la modularisation implique de diviser la vérification de systèmes complexes en sous-problèmes plus petits. La détection de symétrie implique de minimiser l'espace d'état en identifiant les symétries qui se produisent lors de l'exécution du système et en créant une correspondance entre les états et les représentants de classe d'équivalence. Ces techniques ont été utilisées dans des recherches antérieures pour résoudre le problème d'explosion d'état dans la vérification et la validation formelle des systèmes IoT. Cependant, il est encore nécessaire de développer des techniques plus efficaces et plus performantes qui peuvent gérer la nature dynamique et hétérogène des systèmes IoT.

Un autre défi dans la vérification et la validation formelles des systèmes IoT est la nécessité de s'assurer qu'ils répondent aux exigences de performance tout en maintenant la sécurité et la fiabilité. De nombreux systèmes IoT sont utilisés dans des applications critiques pour la sécurité, telles que les soins de santé et les transports, où la fiabilité et la sécurité sont d'une importance primordiale. De plus, les systèmes IoT impliquent souvent des contraintes en temps réel, ce qui peut rendre difficile de s'assurer qu'ils répondent aux exigences de performance. Cela peut être résolu en utilisant des automates temporisés et d'autres modèles formels qui capturent le comportement temporel des systèmes IoT. Cependant, il est encore nécessaire de développer des modèles et des techniques plus sophistiqués qui peuvent gérer les interactions et les dépendances complexes qui existent dans les systèmes IoT. De plus, il est nécessaire de garantir que les techniques de vérification et de validation formellessont intégrées dans le processus de développement logiciel des systèmes IoT, plutôt que d'être considérées comme une réflexion après coup. Cela nécessite un changement culturel vers une approche plus formalisée du développement logiciel, ainsi que le développement d'outils et de cadres qui facilitent l'application des techniques de vérification et de validation formelles.

Ainsi, les techniques de vérification et de validation formelles offrent une approche prometteuse pour garantir la fiabilité et la sécurité des systèmes IoT. Cependant, il existe plusieurs défis et enjeux ouverts qui doivent être résolus pour réaliser pleinement leur potentiel. Ces défis comprennent la complexité et l'hétérogénéité des systèmes IoT, la nature dynamique des systèmes IoT, le problème d'explosion d'état et la nécessité de garantir que les systèmes IoT répondent aux exigences de performance tout en mainte-

nant la sécurité et la fiabilité. Pour relever ces défis, il est nécessaire de développer des techniques plus efficaces et plus sophistiquées, ainsi que de favoriser un changement culturel vers une approche plus formalisée du développement logiciel des systèmes IoT.

7 Conclusion et travaux futurs

En conclusion, les techniques de vérification et de validation formelles ont le potentiel de relever les défis de fiabilité et de sécurité dans les systèmes IoT. Cependant, la nature dynamique et hétérogène des systèmes IoT présente plusieurs défis pour l'application de ces techniques. Les chercheurs ont développé diverses techniques, telles que l'abstraction, la modularisation, la détection de symétrie et les automates temporisés, pour relever ces défis. Cependant, il est encore nécessaire de développer des techniques plus efficaces et plus performantes qui peuvent gérer les interactions et les dépendances complexes qui existent dans les systèmes IoT. De plus, il est nécessaire de favoriser un changement culturel vers une approche plus formalisée du développement logiciel, où les techniques de vérification et de validation formelles sont intégrées dans le processus de développement.

Une direction possible pour les recherches futures est le développement de modèles formels et de techniques de vérification plus sophistiqués qui peuvent gérer la nature dynamique et hétérogène des systèmes IoT. Par exemple, les chercheurs pourraient développer des modèles qui capturent les interactions et les dépendances entre les dispositifs et les réseaux, ainsi que le contexte etl'environnement dans lesquels le système fonctionne. Ils pourraient également développer des techniques de vérification qui peuvent gérer le grand nombre d'états et d'événements qui se produisent dans les systèmes IoT, tout en maintenant les performances et l'évolutivité. Une autre direction possible est le développement d'outils et de cadres qui facilitent l'application des techniques de vérification et de validation formelles dans le processus de développement. Ces outils pourraient automatiser le processus de génération et de vérification de modèles, réduisant l'effort manuel requis et améliorant la précision et la fiabilité des modèles.

De plus, il est nécessaire de développer des normes plus complètes pour les dispositifs et les réseaux IoT qui peuvent améliorer la précision et la fiabilité des modèles formels. La normalisation peut également permettre l'interopérabilité entre les dispositifs et les réseaux, améliorant l'évolutivité et la flexibilité des systèmes IoT. Enfin, il est nécessaire d'étudier l'utilisation de techniques d'apprentissage automatique et d'intelligence artificielle en conjonction avec les techniques de vérification et de validation formelles. Ces techniques peuvent aider à identifier les modèles et les anomalies dans les systèmesIoT, améliorant ainsi leur fiabilité et leur sécurité.

En résumé, les techniques de vérification et de validation formelles offrent une approche prometteuse pour garantir la fiabilité et la sécurité des systèmes IoT. La résolution des défis et des enjeux ouverts abordés dans cet article nécessitera un effort concerté de la part des chercheurs, des développeurs et des acteurs de l'industrie. Avec la poursuite de la recherche et du développement, les techniques de vérification et de validation formelles peuvent contribuer à réaliser le plein potentiel des systèmes IoT de manière sûre et sécurisée.

Références

[1] Qasem Abu Al-Haija, Moez Krichen, and Wejdan Abu Elhaija. Machine-learning-based darknet traffic detection system for iot applications. *Electronics*, 11(4) :556, 2022.

[2] Tanwir Ahmad, Junaid Iqbal, Adnan Ashraf, Dragos Truscan, and Ivan Porres. Model-based testing using uml activity diagrams : A systematic mapping study. *Computer Science Review*, 33 :98–112, 2019.

[3] Abdullah Al Farooq, Ehab Al-Shaer, Thomas Moyer, and Krishna Kant. Iotc 2 : A formal method approach for detecting conflicts in large scale iot systems. In *2019 IFIP/IEEE symposium on integrated network and service management (IM)*, pages 442–447. IEEE, 2019.

[4] Maryam Alamer and Mohammed Amin Almaiah. Cybersecurity in smart city : A systematic mapping study. In *2021 International Conference on Information Technology (ICIT)*, pages 719–724. IEEE, 2021.

[5] Amira Ali, Huda Amin Maghawry, and Nagwa Badr. Performance testing as a service using cloud computing environment : A survey. *Journal of Software : Evolution and Process*, page e2492, 2022.

[6] Manos Antonakakis, Tim April, Michael Bailey, Matt Bernhard, Elie Bursztein, Jaime Cochran, Zakir Durumeric, J. Alex Halderman, Luca Invernizzi, Michalis Kallitsis, Deepak Kumar, Chaz Lever, Zane Ma, Joshua Mason, Damian Menscher, Chad Seaman, Nick Sullivan, Kurt Thomas, and Yi Zhou. Understanding the mirai botnet. In *26th USENIX Security Symposium (USENIX Security 17)*, pages 1093–1110, Vancouver, BC, 2017. USENIX Association.

[7] Bojana Bakić, Miloš Milić, Ilija Antović, Dušan Savić, and Tatjana Stojanović. 10 years since stuxnet : What have we learned from this mysterious computer software worm ? In *2021 25th International Conference on Information Technology (IT)*, pages 1–4. IEEE, 2021.

[8] Clark Barrett and Cesare Tinelli. Satisfiability modulo theories. In *Handbook of model checking*, pages 305–343. Springer, 2018.

[9] Saddek Bensalem, Moez Krichen, Lotfi Majdoub, Riadh Robbana, and Stavros Tripakis. A simplified approach for testing real-time systems based on action refinement. In *ISoLA*, pages 191–202, 2007.

[10] Jerry R Burch, Roberto Passerone, and Alberto L Sangiovanni-Vincentelli. Modeling techniques in design-by-refinement methodologies. In *System Specification & Design Languages*, pages 283–292. Springer, 2003.

[11] Patrick Cousot. Abstract interpretation based formal methods and future challenges. In *Informatics*, pages 138–156. Springer, 2001.

[12] Jennifer A Davis, Matthew Clark, Darren Cofer, Aaron Fifarek, Jacob Hinchman, Jonathan Hoffman, Brian Hulbert, Steven P Miller, and Lucas Wagner. Study on the barriers to the industrial adoption of formal methods. In *Int. workshop on formal methods for industrial critical systems*, pages 63–77. Springer, 2013.

[13] Steve Easterbrook and John Callahan. Formal methods for verification and vali-

dation of partial specifications : A case study. *Journal of Systems and Software*, 40(3) :199–210, 1998.

[14] Ying Fu, Meng Ren, Fuchen Ma, Heyuan Shi, Xin Yang, Yu Jiang, Huizhong Li, and Xiang Shi. Evmfuzzer : detect evm vulnerabilities via fuzz testing. In *Proceedings of the 2019 27th ACM joint meeting on european software engineering conference and symposium on the foundations of software engineering*, pages 1110–1114, 2019.

[15] Jie Gao and Muthu Ramachandran. A survey on software testing techniques using artificial intelligence. *Journal of Big Data*, 5(1) :1–35, 2018.

[16] Vahid Garousi, Alper Buğra Keleş, Yunus Balaman, Zeynep Özdemir Güler, and Andrea Arcuri. Model-based testing in practice : An experience report from the web applications domain. *Journal of Systems and Software*, 180 :111032, 2021.

[17] Herman Geuvers. Proof assistants : History, ideas and future. *Sadhana*, 34(1) :3–25, 2009.

[18] Mario Gleirscher, Simon Foster, and Jim Woodcock. New opportunities for integrated formal methods. *ACM Computing Surveys (CSUR)*, 52(6) :1–36, 2019.

[19] Mario Gleirscher and Diego Marmsoler. Formal methods in dependable systems engineering : a survey of professionals from europe and north america. *Empirical Software Engineering*, 25(6) :4473–4546, 2020.

[20] Anjana Gosain and Ganga Sharma. Static analysis : A survey of techniques and tools. In *Intelligent Computing and Applications*. Springer, 2015.

[21] Morten Hertzum. Usability testing : A practitioner's guide to evaluating the user experience. *Synthesis Lectures on Human-Centered Informatics*, 13(1) :i–105, 2020.

[22] Katharina Hofer-Schmitz and Branka Stojanović. Towards formal methods of iot application layer protocols. In *2019 12th CMI conference on cybersecurity and privacy (CMI)*, pages 1–6. IEEE, 2019.

[23] Katharina Hofer-Schmitz and Branka Stojanović. Towards formal verification of iot protocols : A review. *Computer Networks*, 174 :107233, 2020.

[24] Rateb Jabbar, Noora Fetais, Mohamed Kharbeche, Moez Krichen, Kamel Barkaoui, and Mohammed Shinoy. Blockchain for the internet of vehicles : how to use blockchain to secure vehicle-to-everything (v2x) communication and payment ? *IEEE Sensors Journal*, 21(14) :15807–15823, 2021.

[25] Rateb Jabbar, Mohammed Shinoy, Mohamed Kharbeche, Khalifa Al-Khalifa, Moez Krichen, and Kamel Barkaoui. Urban traffic monitoring and modeling system : An iot solution for enhancing road safety. In *2019 international conference on internet of things, embedded systems and communications (iintec)*, pages 13–18. IEEE, 2019.

[26] Bryer Jeannotte and Ali Tekeoglu. Artorias : Iot security testing framework. In *2019 26th International Conference on Telecommunications (ICT)*, pages 233–237. IEEE, 2019.

[27] K Keerthi, Indrani Roy, Aritra Hazra, and Chester Rebeiro. Formal verification for security in iot devices. *Security and Fault Tolerance in Internet of Things*,

pages 179–200, 2019.

[28] Vladimir Khorikov. *Unit Testing Principles, Practices, and Patterns*. Simon and Schuster, 2020.

[29] Jonis Kiesbye, David Messmann, Maximilian Preisinger, Gonzalo Reina, Daniel Nagy, Florian Schummer, Martin Mostad, Tejas Kale, and Martin Langer. Hardware-in-the-loop and software-in-the-loop testing of the move-ii cubesat. *Aerospace*, 6(12) :130, 2019.

[30] Moez Krichen. *Model-based testing for real-time systems*. PhD thesis, PhD thesis, PhD thesis, Universit Joseph Fourier (December 2007), 2007.

[31] Moez Krichen. A formal framework for conformance testing of distributed real-time systems. In *International Conference On Principles Of Distributed Systems*, pages 139–142. Springer, 2010.

[32] Moez Krichen. A formal framework for black-box conformance testing of distributed real-time systems. *International Journal of Critical Computer-Based Systems*, 3(1-2) :26–43, 2012.

[33] Moez Krichen. *Contributions to model-based testing of dynamic and distributed real-time systems*. PhD thesis, École Nationale d'Ingénieurs de Sfax (Tunisie), 2018.

[34] Moez Krichen. Improving formal verification and testing techniques for internet of things and smart cities. *Mobile networks and applications*, pages 1–12, 2019.

[35] Moez Krichen, Omar Cheikhrouhou, Mariam Lahami, Roobaea Alroobaea, and Afef Jmal Maâlej. Towards a model-based testing framework for the security of internet of things for smart city applications. In *Smart Societies, Infrastructure, Technologies and Applications : First International Conference, SCITA 2017, Jeddah, Saudi Arabia, November 27–29, 2017, Proceedings 1*, pages 360–365. Springer International Publishing, 2018.

[36] Moez Krichen, Afef Jmal Maâlej, and Mariam Lahami. A model-based approach to combine conformance and load tests : an ehealth case study. *International Journal of Critical Computer-Based Systems*, 8(3-4) :282–310, 2018.

[37] Moez Krichen, Seifeddine Mechti, Roobaea Alroobaea, Elyes Said, Parminder Singh, Osamah Ibrahim Khalaf, and Mehedi Masud. A formal testing model for operating room control system using internet of things. *Computers, Materials & Continua*, 66(3) :2997–3011, 2021.

[38] Moez Krichen and Stavros Tripakis. State identification problems for timed automata. In *TestCom*, volume 5, pages 175–191, 2005.

[39] Moez Krichen and Stavros Tripakis. Interesting properties of the real-time conformance relation tioco. In *Theoretical Aspects of Computing-ICTAC 2006 : Third International Colloquium, Tunis, Tunisia, November 20-24, 2006. Proceedings 3*, pages 317–331. Springer Berlin Heidelberg, 2006.

[40] Asif Ali Laghari, Kaishan Wu, Rashid Ali Laghari, Mureed Ali, and Abdullah Ayub Khan. A review and state of art of internet of things (iot). *Archives of Computational Methods in Engineering*, pages 1–19, 2021.

[41] Mariam Lahami and Moez Krichen. A survey on runtime testing of dynamically adaptable and distributed systems. *Software Quality Journal*, 29(2) :555–593, 2021.

[42] Mariam Lahami, Moez Krichen, Hajer Barhoumi, and Mohamed Jmaiel. Selective test generation approach for testing dynamic behavioral adaptations. In *IFIP International Conference on Testing Software and Systems*, pages 224–239. Springer, Cham, 2015.

[43] Mariam Lahami, Moez Krichen, Mariam Bouchakwa, and Mohamed Jmaiel. Using knapsack problem model to design a resource aware test architecture for adaptable and distributed systems. In *Testing Software and Systems : 24th IFIP WG 6.1 International Conference, ICTSS 2012, Aalborg, Denmark, November 19-21, 2012. Proceedings 24*, pages 103–118. Springer Berlin Heidelberg, 2012.

[44] Mariam Lahami, Moez Krichen, and Mohamed Jmaïel. Runtime testing approach of structural adaptations for dynamic and distributed systems. *International Journal of Computer Applications in Technology*, 51(4) :259–272, 2015.

[45] In Lee. Internet of things (iot) cybersecurity : Literature review and iot cyber risk management. *Future Internet*, 12(9) :157, 2020.

[46] Afef Jmal Maâlej, Manel Hamza, Moez Krichen, and Mohamed Jmaiel. Automated significant load testing for ws-bpel compositions. In *2013 IEEE sixth international conference on software testing, verification and validation workshops*, pages 144–153. IEEE, 2013.

[47] Afef Jmal Maâlej and Moez Krichen. A model based approach to combine load and functional tests for service oriented architectures. In *VECoS*, pages 123–140, 2016.

[48] Afef Jmal Maâlej, Moez Krichen, and Mohamed Jmaiel. Conformance testing of ws-bpel compositions under various load conditions. In *2012 IEEE 36th annual computer software and applications conference*, pages 371–371. IEEE, 2012.

[49] Afef Jmal Maâlej, Moez Krichen, and Mohamed Jmaiel. Model-based conformance testing of ws-bpel compositions. In *2012 IEEE 36th annual computer software and applications conference workshops*, pages 452–457. IEEE, 2012.

[50] Afef Jmal Maâlej, Mariam Lahami, Moez Krichen, and Mohamed Jmaïel. Distributed and resource-aware load testing of ws-bpel compositions. In *ICEIS (2)*, pages 29–38, 2018.

[51] Sara N Matheu-García, José L Hernández-Ramos, Antonio F Skarmeta, and Gianmarco Baldini. Risk-based automated assessment and testing for the cybersecurity certification and labelling of iot devices. *Computer Standards & Interfaces*, 62 :64–83, 2019.

[52] Franc Mihalič, Mitja Truntič, and Alenka Hren. Hardware-in-the-loop simulations : A historical overview of engineering challenges. *Electronics*, 11(15) :2462, 2022.

[53] Alaeddine Mihoub, Ouissem Ben Fredj, Omar Cheikhrouhou, Abdelouahid Derhab, and Moez Krichen. Denial of service attack detection and mitigation for

internet of things using looking-back-enabled machine learning techniques. *Computers & Electrical Engineering*, 98 :107716, 2022.

[54] Barton P Miller, Mengxiao Zhang, and Elisa R Heymann. The relevance of classic fuzz testing : Have we solved this one ? *IEEE Transactions on Software Engineering*, 48(6) :2028–2039, 2020.

[55] Markus Müller-Olm, David Schmidt, and Bernhard Steffen. Model-checking. In *International Static Analysis Symposium*, pages 330–354. Springer, 1999.

[56] Amir Pnueli, Sitvanit Ruah, and Lenore Zuck. Automatic deductive verification with invisible invariants. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 82–97. Springer, 2001.

[57] S Phani Shashank, Praneeth Chakka, and D Vijay Kumar. A systematic literature survey of integration testing in component-based software engineering. In *2010 International Conference on Computer and Communication Technology (ICCCT)*, pages 562–568. IEEE, 2010.

[58] Shachar Siboni, Vinay Sachidananda, Yair Meidan, Michael Bohadana, Yael Mathov, Suhas Bhairav, Asaf Shabtai, and Yuval Elovici. Security testbed for internet-of-things devices. *IEEE transactions on reliability*, 68(1) :23–44, 2019.

[59] Alireza Souri and Monire Norouzi. A state-of-the-art survey on formal verification of the internet of things applications. *Journal of Service Science Research*, 11(1) :47–67, 2019.

[60] Jing Tian, Yuanfang Li, and Xiaoyuan Zhang. A survey on software testing with machine learning. *Journal of Software : Evolution and Process*, 31(7) :e2176, 2019.

[61] Porfirio Tramontana, Domenico Amalfitano, Nicola Amatucci, and Anna Rita Fasolino. Automated functional testing of mobile applications : a systematic mapping study. *Software Quality Journal*, 27(1) :149–201, 2019.

[62] Jack van Heugten Breurkes, Fabian Gilson, and Matthias Galster. Overlap between automated unit and acceptance testing–a systematic literature review. In *Proceedings of the International Conference on Evaluation and Assessment in Software Engineering 2022*, pages 80–89, 2022.

[63] Juan Pablo Vielma. Mixed integer linear programming formulation techniques. *Siam Review*, 57(1) :3–57, 2015.

[64] Fujun Wang, Zining Cao, Lixing Tan, and Hui Zong. Survey on learning-based formal methods : Taxonomy, applications and possible future directions. *IEEE Access*, 8 :108561–108578, 2020.

[65] Bin Xie, Shuai Wang, Xiuheng Wu, Changkai Wen, Shengli Zhang, and Xueyan Zhao. Design and hardware-in-the-loop test of a coupled drive system for electric tractor. *Biosystems Engineering*, 216 :165–185, 2022.