Jupiter: Publikasi Ilmu Keteknikan Industri, Teknik Elektro dan Informatika
Vol.2, No.2 Maret 2024
e-ISSN: 3031-349X; p-ISSN: 3031-500X, Hal 181-191
DOI: https://doi.org/10.61132/jupiter.v2i2.155

Analisis Perbandingan Implementasi Clean Architecture Menggunakan MVP, MVI, Dan MVVM Pada Pengembangan Aplikasi Android Native

Firmansyah Firdaus Anhar
Universitas Pembangunan Nasional "Veteran" Jawa Timur

Made Hanindia Prami Swari
Universitas Pembangunan Nasional "Veteran" Jawa Timur

Firza Prima Aditiawan
Universitas Pembangunan Nasional "Veteran" Jawa Timur

Alamat: Jl.Raya Rungkut Madya, Gunung Anyar, Surabaya, Indonesia
*Korespondensi penulis: fafirmansyah20@gmail.com**

Abstract*. Clean architecture is a method of application development that divides code into multiple layers based on the purpose of the code, ensuring minimal dependency. Popular architectures in Android application development include MVP (Model View Presenter), MVI (Model View Intent), and MVVM (Model View ViewModel). This research focuses on creating three applications with similar features and interfaces using different architectures. The study compares modifiability, testability, and performance aspects to determine the differences between each architecture. The results show that MVVM architecture is the best in modifiability, with the lowest number of index modifications. Testability requires no more than four scenarios for all architectures. However, MVI architecture outperforms in test coverage, and MVP architecture outperforms in performance. Overall, clean architecture is a valuable approach for improving the performance and usability of Android applications.*

Keywords*: Android, Clean Architecture, MVI, MVP, MVVM.*

Abstrak. *Clean arsitektur* adalah sebuah konsep atau metode pengembangan aplikasi yang berfokus pada pemisahan bagian kode tertentu yang terbagi dalam beberapa lapis kode berdasarkan tujuan dari kode tersebut, sehingga satu bagian dan bagian lain sedikit atau bahkan tidak memiliki ketergantungan. Beberapa arsitektur populer yang sering digunakan dalam pengembangan aplikasi Android yang menjadi fokus utama dalam penelitian ini yaitu MVP (Model View Presenter), MVI (Model View Intent), dan MVVM (Model View ViewModel). Melalui penelitian ini peneliti membuat sebuah 3 aplikasi berbeda dengan fitur dan antarmuka yang sama dengan arsitektur yang berbeda. Masing masing aplikasi dibandingkan berdasarkan aspek modifiabilitas, testabilitas, dan performa untuk mendapatkan perbedaan masing masing arsitektur dengan akurat. Hasil perbandingan menunjukkan, dalam aspek modifiabilitas, arsitektur MVVM menjadi arsitektur terbaik dengan jumlah indeks modifiabilitas paling rendah dengan jumlah file yang harus dibuat sebanyak 2 file, kelas yang harus dibuat sebanyak 2 kelas, dan fungsi yang harus dibuat sebanyak 5 fungsi. Dalam aspek testabilitas, semua arsitektur memiliki jumlah skenario yang sama yaitu 4 skenario, tidak ada satu arsitektur yang memerlukan lebih dari 4 skenario. Namun dalam bagian test coverage, arsitektur MVI menjadi arsitektur dengan tingkat testabilitas paling baik. Dalam aspek performa, arsitektur MVP menjadi arsitektur dengan hasil terbaik dibandingkan dengan arsitektur lainnya.

Kata kunci: Android, Clean Architecture, MVI, MVP, MVVM.

Received Desember 31, 2023; Accepted Januari 18, 2024; Published Maret 31, 2024
* Firmansyah Firdaus Anhar, *fafirmansyah20@gmail.com*

LATAR BELAKANG

Clean architecture adalah sebuah konsep atau metode pengembangan aplikasi yang berfokus pada pemisahan bagian kode tertentu yang terbagi dalam beberapa lapis kode berdasarkan tujuan dari kode tersebut, sehingga satu bagian dan bagian lain sedikit atau bahkan tidak memiliki ketergantungan (Robert C. Martin, 2012).

Pengembangan aplikasi android sudah dilakukan cukup lama, dalam pengembangannya tercipta berbagai macam model arsitektur yang dapat diterapkan. Sistem operasi android sudah semakin berkembang sampai pada titik di mana diperlukan sebuah arsitektur baru yang dapat mendukung kinerja sistem operasi android dengan baik. Beberapa arsitektur populer yang sering digunakan dalam pengembangan aplikasi Android yang menjadi fokus utama dalam penelitian ini yaitu MVP (Model View Presenter), MVI (Model View Intent), dan MVVM (Model View ViewModel).

Penelitian terkait telah dilakukan dengan membandingkan jenis arsitektur yang cukup berbeda, yaitu MVC, MVP dan MVVM yang ditulis pada tesis berjudul "A comparison of Android native app architecture MVC, MVP and MVVM" oleh Lou T. pada tahun 2016. Perbandingan dilakukan dengan beberapa jenis parameter, seperti modifiabilitas, testabilitas, dan performa berdasarkan skenario tertentu untuk menentukan arsitektur mana yang paling sesuai dengan jenis pengembangan aplikasi tertentu.

KAJIAN TEORITIS

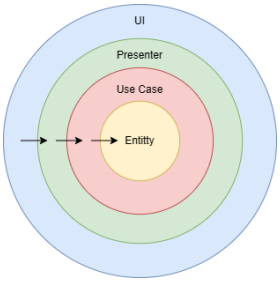
Penelitian Terdahulu

A comparison of Android native app architecture MVC, MVP and MVVM. Penelitian ini ditulis oleh Lou T. pada tahun 2016, menjelaskan komparasi tiga arsitektur pada Android, yaitu MVC, MVP, dan MVVM. Hasil dari penelitian ini adalah arsitektur MVVM memiliki keunggulan testabilitas, MVC memiliki keunggulan modifiabilitas, MVP memiliki keunggulan performa.

Performance Comparison of Native Android Application on MVP and MVVM. Penelitian ini ditulis oleh Bambang Wisnuadhi, Ghifari Munawar, dan Ujang Wahyu pada tahun 2020, menjelaskan komparasi dua arsitektur pada Android, yaitu MVP dan MVVM. Pada penelitian ini menunjukkan bahwa arsitektur MVVM mendapatkan hasil yang lebih baik daripada arsitektur MVP.

Clean Architecture

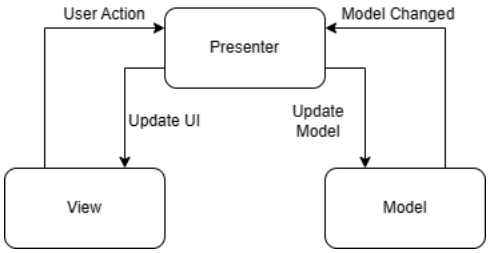
Clean architecture adalah sebuah konsep atau metode pengembangan aplikasi yang berfokus pada pemisahan bagian kode tertentu yang terbagi dalam beberapa lapis kode berdasarkan tujuan dari kode tersebut, sehingga satu bagian dan bagian lain sedikit atau bahkan tidak memiliki ketergantungan (Robert C. Martin, 2012). Konsep *clean architecture* dibuat oleh Robert C. Martin yang ditulis pada blog pribadinya. Berikut adalah ilustrasi dari gambaran umum terkait *clean architecture* yang dilampirkan pada **Gambar 1.**



Gambar 1. Gambaran umum model clean architecture

MVP (Model View Presenter)

MVP adalah singkatan dari Model-View-Presenter merupakan salah satu jenis arsitektur yang sering digunakan pada pengembangan aplikasi Android. Model di sini bertanggung jawab untuk menyimpan objek domain dan melakukan operasi logis pada objek tersebut. View bertanggung jawab untuk membangun user interface. Presenter bertanggung jawab untuk menangani semua panggilan balik dari tampilan dan menggunakan model untuk mendapatkan data, yang diperlukan agar tampilan dapat membangun antarmuka pengguna yang tepat (Humeniuk, V. 2019). Diagram alur sederhana dari arsitektur MVP bisa dilihat pada **Gambar 2**.

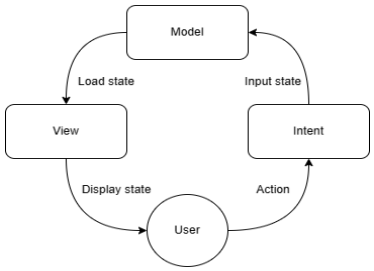


Gambar 2. Alur arsitektur MVP

MVI (Model View Intent)

Arsitektur MVI, merupakan sebuah arsitektur yang menerapkan aliran data searah (*unidirectional flow*). Model adalah keadaan sebuah aplikasi, yang masih dalam bentuk sebuah data mentah. View adalah representasi keadaan sebuah aplikasi (dari model) yang ditampilkan kepada pengguna. Intent adalah sebuah tindakan yang dilakukan sistem untuk merespon

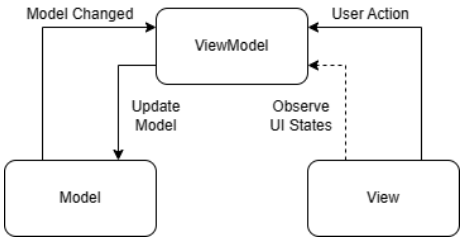
masukan dari pengguna dan perubahan pada keadaan aplikasi (dari model) (Chauhan, K., Kumar, S., Sethia, D., & Alam, M. N. 2021). Diagram alur sederhana dari arsitektur MVI dapat dilihat pada **Gambar 3.**



Gambar 3. Alur arsitektur MVP

MVVM (Model View ViewModel)

Model View ViewModel atau yang biasa disingkat MVVM merupakan salah satu desain arsitektur yang mulai sering digunakan dalam pengembangan aplikasi mobile khususnya android. MVVM memiliki beberapa tujuan, seperti mempermudah dalam penulisan kode dan melakukan testing, hingga bisa bertahan dalam perubahan konfigurasi pada aplikasi. MVVM merupakan hasil evolusi dari arsitektur sebelumnya yaitu MVP. Dalam evolusinya, MVVM bertujuan untuk membuat arsitektur menjadi lebih terpisah antara antarmuka dengan domain layer-nya (Krasko, R. 2020). Diagram alur sederhana dari arsitektur MVVM dapat dilihat pada **Gambar 4.**

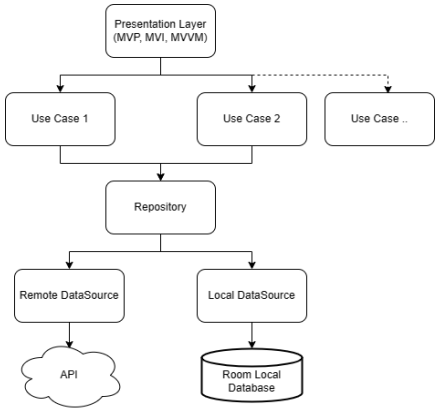


Gambar 3. Alur arsitektur MVVM

METODE PENELITIAN

Perancangan Sistem

Perancangan sistem dilakukan dengan membuat struktur dasar untuk *data layer* menggunakan bagan alur untuk memudahkan pengembangan sistem. Selanjutnya dilakukan perancangan *template* dasar untuk masing masing jenis arsitektur sehingga proses pengembangan sistem dapat menjadi lebih singkat dan pengukuran perbandingan menjadi lebih akurat. Alur sistem dasar yang digunakan pada semua arsitektur dapat dilihat pada **Gambar 4.**



Gambar 4. Alur data layer sistem aplikasi

Kriteria Perbandingan

1. Modifiabilitas

Modifiabilitas adalah bagaimana sebuah arsitektur dapat mudah dilakukan perubahan atau penambahan pada fitur. Untuk mengukur modifiabilitas sebuah arsitektur dilakukan dengan menghitung sumber daya yang diperlukan untuk melakukan perubahan atau penambahan fitur pada aplikasi.

2. Testabilitas

Testabilitas mengacu pada seberapa mudah dan tepat suatu arsitektur dapat diuji. Untuk mengukur testabilitas suatu arsitektur dapat juga dilakukan dengan melihat *test coverage* dari masing masing arsitektur, *test coverage* merupakan cakupan kode yang telah diuji, sehingga semakin tinggi nilai *coverage*-nya maka sebuah sistem semakin teruji.

3. Performa

Perbedaan performa pada aplikasi dapat memengaruhi pengalaman pengguna dalam menggunakan aplikasi tersebut. Pengukuran performa dilakukan dengan melihat penggunaan memori dan CPU pada aplikasi, juga dilakukan dengan mengukur waktu yang diperlukan untuk aplikasi dibuka, serta menghitung parameter lain seperti *high input latency, slow UI thread, dan slow draw command*.

Pengujian Sistem

Pengujian dilakukan dengan beberapa skenario yang sama pada masing masing aplikasi untuk mendapat hasil yang akurat pada masing masing jenis arsitektur. Adapun beberapa jenis *tools* atau metode yang digunakan pada pengujian sistem ini, yaitu JUnit sebagai kerangka kerja untuk penulisan kode *unit testing,* Android Profiler digunakan untuk melakukan *monitoring* penggunaan sumber daya aplikasi secara *real-time* untuk melihat performa aplikasi yang sedang dikembangkan, AndroidX Benchmark digunakan untuk melakukan proses *macro benchmarking* dengan tujuan untuk mengetahui performa aplikasi yang sedang dibuat. Firebase

Test Lab digunakan untuk melakukan pengujian performa aplikasi menggunakan berbagai macam perangkat nyata yang disediakan secara *cloud* oleh Firebase.

Lingkungan Pengujian dan Pengembangan Sistem

Untuk meningkatkan akurasi dan ketepatan sebuah pengujian dan perbandingan arsitektur, diperlukan sebuah perangkat dasar sama yang digunakan untuk masing masing jenis arsitektur. Perangkat yang digunakan adalah Realme 10, Google Pixel 5, Google Pixel 6, dan Samsung Galaxy Z Fold 3.

HASIL DAN PEMBAHASAN

Modifiabilitas

Hasil perbandingan dari modifiabilitas masing masing arsitektur dilakukan dengan mencari arsitektur dengan nilai modifiabilitas paling rendah. Hasil dari perbandingan tingkat modifiabilitas dari masing masing arsitektur dapat dilihat pada **Tabel 1.**

Tabel 1. Perbandingan modifiabilitas

Arsitektur	File Baru	Kelas Baru	Fungsi Baru
MVVM	2	2	5
MVP	2	2	7
MVI	3	3	7

Testabilitas

Jumlah Skenario Uji Coba

Tabel 2. Perbandingan jumlah skenario uji coba

Arsitektur	Jumlah Skenario Uji Coba
MVVM	4
MVP	4
MVI	4

Tabel 2 menunjukkan bahwa pada masing masing arsitektur memiliki jumlah skenario uji coba yang sama, yaitu empat skenario. Tidak ada perbedaan dalam jumlah skenario yang perlu dilakukan pada masing masing arsitektur.

Test Coverage

Perbandingan *test coverage* dilakukan dengan membandingkan hasil dari *test coverage* untuk masing masing arsitektur dengan pengelompokkan berdasarkan kelas, fungsi, dan baris kode yang telah masuk dalam *test coverage*.

***a) Test Coverage* Dalam Kelas**

Tabel 3. Perbandingan test coverage untuk kelas

Test Target	*Coverage* MVVM	*Coverage* MVP	*Coverage* MVI
Project	19% (164/860)	20% (172/860)	19% (172/904)
ViewModel/ Presenter	81% (26/32)	82% (28/34)	72% (24/33)
Repository	100% (8/8)	100% (8/8)	100% (8/8)
Resource/ Model	75% (3/4)	75% (3/4)	87% (28/32)

Berdasarkan perbandingan pada **Tabel 3**, secara umum arsitektur MVI memiliki nilai *coverage* paling besar di antara arsitektur lain walaupun dengan satu target uji yang memiliki nilai lebih rendah dari arsitektur lain, sehingga MVI bisa dibilang sebagai arsitektur dengan nilai *coverage* terbaik pada skala kelas.

***b) Test Coverage* Dalam Fungsi**

Tabel 4. Perbandingan test coverage untuk fungsi

Test Target	*Coverage* MVVM	*Coverage* MVP	*Coverage* MVI
Project	23% (76/320)	20% (64/320)	27% (96/352)
ViewModel/ Presenter	76% (10/13)	70% (7/10)	68% (11/16)
Repository	100% (8/8)	100% (8/8)	100% (8/8)
Resource/ Model	75% (3/4)	75% (3/4)	87% (28/32)

Berdasarkan perbandingan pada Tabel 4, secara umum arsitektur MVI memiliki nilai coverage paling besar di antara arsitektur lain walaupun dengan satu target uji yang memiliki nilai lebih rendah dari arsitektur lain, sehingga MVI bisa dibilang sebagai arsitektur dengan nilai coverage terbaik pada skala fungsi.

a) *Test Coverage* Dalam Baris Kode

Tabel 5. Perbandingan *test coverage* untuk bari kode

Test Target	*Coverage* MVVM	*Coverage* MVP	*Coverage* MVI
Project	19% (164/860)	20% (172/860)	19% (172/904)
ViewModel/ Presenter	81% (26/32)	82% (28/34)	72% (24/33)
Repository	100% (8/8)	100% (8/8)	100% (8/8)
Resource/ Model	75% (3/4)	75% (3/4)	87% (28/32)

Berdasarkan perbandingan **Tabel 5**, secara umum arsitektur MVP memiliki nilai *coverage* paling besar di antara arsitektur lain walaupun dengan satu target uji yang memiliki nilai lebih rendah dari arsitektur lain, sehingga MVP bisa dibilang sebagai arsitektur dengan nilai *coverage* terbaik pada skala baris kode.

Performa

1. Firebase Test Lab

a) Google Pixel 5

Tabel 6. Perbandingan Firebase Test Lab pada Google Pixel 5

Arsitektur	*Time to initial display*	*High Input Latency*	*Slow UI Thread*	Maksimal CPU	Maksimal Memori
MVVM	527ms	6%	1%	18.71%	223.768KiB
MVP	634ms	4%	1%	15.625%	229.048KiB
MVI	557ms	9%	1%	16.774%	214.448KiB

Tabel 6 menunjukkan bahwa arsitektur MVP memiliki nilai *time to initial display* terlama, yaitu 634ms, yang artinya arsitektur ini memiliki waktu terlama dalam membuka aplikasi. *High input latency* memiliki nilai tertinggi pada arsitektur MVI sebesar 9% yang artinya pada arsitektur ini, proses input yang lambat lebih sering terjadi daripada arsitektur

lainnya. Nilai *slow UI thread* pada semua arsitektur menunjukkan nilai yang sama yaitu 1%, artinya tidak ada perbedaan dalam aspek ini untuk masing masing arsitektur. Maksimal penggunaan CPU tertinggi dimiliki oleh arsitektur MVVM sebesar 18.71%, namun penggunaan maksimal dari memori dimiliki oleh arsitektur MVP sebesar 229 MB.

b) Google Pixel 6

Tabel 7. Perbandingan Firebase Test Lab pada Google Pixel 6

Arsitektur	*Time to initial display*	*High Input Latency*	*Slow UI Thread*	Maksimal CPU	Maksimal Memori
MVVM	217ms	12%	1%	15.19%	223.316KiB
MVP	145ms	28%	1%	13.462%	172.368KiB
MVI	243ms	14%	1%	20.253%	217.952KiB

Tabel 7 menunjukkan bahwa arsitektur MVI memiliki nilai *time to initial display* terlama, yaitu 243ms, yang artinya arsitektur ini memiliki waktu terlama dalam membuka aplikasi. *High input latency* memiliki nilai tertinggi pada arsitektur MVP sebesar 28% yang artinya pada arsitektur ini, proses input yang lambat lebih sering terjadi daripada arsitektur lainnya. Nilai *slow UI thread* pada semua arsitektur menunjukkan nilai yang sama yaitu 1%, artinya tidak ada perbedaan dalam aspek ini untuk masing masing arsitektur. Maksimal penggunaan CPU tertinggi dimiliki oleh arsitektur MVI sebesar 20.253%, namun penggunaan maksimal dari memori dimiliki oleh arsitektur MVVM sebesar 223MB.

c) Samsung Galaxy Z Fold 3

Tabel 8. Perbandingan Firebase Test Lab pada Samsung Galaxy Z Fold 3

Arsitektur	*Time to initial display*	*High Input Latency*	*Slow UI Thread*	Maksimal CPU	Maksimal Memori
MVVM	534ms	58%	5%	15.625%	167.92KiB
MVP	523ms	56%	6%	13.291%	159.324KiB
MVI	633ms	118%	7%	14.103%	174.944KiB

Tabel 8 menunjukkan bahwa arsitektur MVI memiliki nilai *time to initial display* terlama, yaitu 633ms, yang artinya arsitektur ini memiliki waktu terlama dalam membuka aplikasi. *High input latency* memiliki nilai tertinggi pada arsitektur MVI sebesar 118%, lebih dari dua kali lipat dari arsitektur lainnya, yang artinya pada arsitektur ini, proses input yang lambat lebih sering terjadi daripada arsitektur lainnya. Nilai *slow UI thread* pada arsitektur ini memiliki nilai tertinggi pada arsitektur MVI sebesar 7%. Maksimal penggunaan CPU tertinggi dimiliki oleh arsitektur MVVM sebesar 15.625% meskipun tidak ada perbedaan yang signifikan antara arsitektur lain, namun penggunaan maksimal dari memori dimiliki oleh arsitektur MVI sebesar 174 MB.

2. Android Profiler

Tabel 9.Perbandingan hasil Android Profiler pada perangkat Realme 10

Arsitektur	Maksimal CPU	Maksimal Memori
MVVM	13%	141MB
MVP	12%	113MB
MVI	12%	106MB

Pada **Tabel 9** Arsitektur MVVM memiliki nilai maksimal CPU tertinggi, artinya pada arsitektur MVVM memerlukan proses CPU yang lebih tinggi. Nilai maksimal dari penggunaan memori juga dihasilkan dari arsitektur MVVM sebesar 141 MB, cukup berbeda jauh dengan arsitektur lainnya pada angka 113 MB dan 106 MB. Proses perbandingan dari penggunaan Android Profiler secara keseluruhan menunjukkan bahwa arsitektur MVVM merupakan arsitektur dengan performa terburuk dibandingkan dengan kedua arsitektur lainnya.

3. Android Macrobenchmark

Tabel 10. Perbandingan hasil benchmarking pada perangkat Realme 10

Arsitektur	Avg Min	Avg Max	Avg Median
MVVM	679	958.65	823.88
MVP	646.74	968.40	753.86
MVI	705.22	1,007.34	786.24

Tabel 10 menunjukkan bahwa arsitektur MVI memiliki nilai rata rata minimal dan maksimal yang paling tinggi dibandingkan dengan arsitektur lainnya, namun untuk rata rata median, arsitektur MVVM memiliki nilai paling tinggi.

KESIMPULAN DAN SARAN

Dari ketiga aspek yang telah diuji, arsitektur MVVM memiliki keunggulan dalam modifiabilitas. Arsitektur MVP memiliki keunggulan dalam hal performa. Arsitektur MVI memiliki keunggulan dalam hal testabilitas. Berdasarkan penelitian ini, maka jika ingin membangun sebuah sistem dengan tingkat modifiabilitas yang bagus, maka MVVM menjadi pilihan arsitektur terbaik. Jika ingin membangun sebuah sistem dengan performa yang baik, maka arsitektur MVP menjadi pilihan arsitektur terbaik. Sedangkan jika ingin membuat sistem dengan tingkat *coverage* yang baik, maka arsitektur MVI bisa digunakan sebagai pilihan. Saran yang dapat digunakan sebagai acuan pada penelitian berikutnya adalah dengan melakukan perbandingan lebih spesifik pada aspek modifiabilitas, testabilitas, dan performa yang akan dibandingkan sehingga didapat hasil yang lebih mendetail, misalnya dengan menghitung rata rata penggunaan memori dan CPU dalam satuan waktu tertentu.

DAFTAR REFERENSI

Lou, T. (2016). A comparison of Android native app architecture MVC, MVP and MVVM. Master's Thesis, Eindhoven: Eindhoven University of Technology.

Krasko, R., & Oskin, A. (2020). Clean Architecture For Android Applications. MATERIALS OF XII JUNIOR RESEARCHERS' CONFERENCE, UDC 004.41.

Humeniuk, V. (2019). Android Architecture Comparison: MVP vs. VIPER.Robert C. Martin. (2021). The Clean Architecture. Available at:https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html,diakses tanggal 20 Desember 2023.

Chauhan, K., Kumar, S., Sethia, D., & Alam, M. N. (2021, May). Performance Analysis of Kotlin Coroutines on Android in a Model-View-Intent Architecture pattern. In 2021 2nd International Conference for Emerging Technology (INCET) (pp. 1-6). IEEE.