BAB II

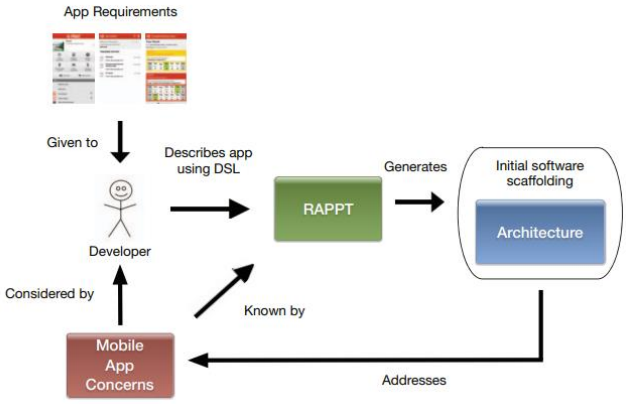
TINJAUAN PUSTAKA

2.1 Penelitian Terdahulu

Penelitian dengan tujuan membuat sistem otomasi untuk menulis kode sebelumnya telah cukup banyak dilakukan dengan berbagai metode yang beragam. Diantaranya adalah sebagai berikut.

2.1.1 Bootstrapping Mobile App Development[2]

Barnett, Scott; Vasa, Rajesh; Grundy, John; dalam penelitian berjudul *Bootstrapping Mobile App Development* merumuskan RAPPT2, yaitu metode otomasi *development* untuk pengembangan aplikasi *mobile* berbasis android yang terinspirasi dari teknik *Model Driven Development* (MDD). RAPPT2 dirancang untuk membantu *developer* pada proses pengembangan aplikasi *mobile*. Selama proses pengembangan, *developer* seringkali melakukan repetisi untuk membuat *block* UI ataupun fungsionalitas kode yang sama atau mirip, karena aplikasi berbasis data pada umumnya memiliki tampilan yang sama namun berbeda data yang ditampilkan. RAPPT dirancang untuk dapat membantu *developer* dengan generasi kode berdasarkan *codebase* atau *boilerplate* yang dihasilkan sehingga terdapat fleksibilitas untuk membuat aplikasi dengan kualitas yang baik. Tertera pada Gambar 2.1 bahwa RAPPT membantu *developer* untuk membuat *starter scaffolding code* dengan hanya DSL kode yang mendeskripsikan *requirement* aplikasi yang dibuat tanpa harus menuliskan kode dengan cara tradisional.



Gambar 2.1 Alur penggunaan RAPPT yang bertujuan untuk membantu *developer* dalam mengembangkan aplikasi *mobile* [2]

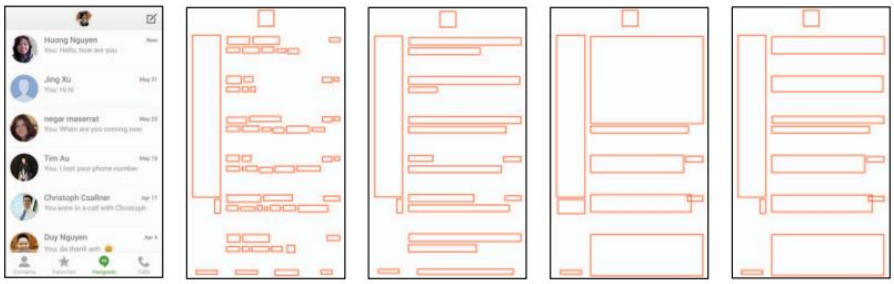
Disebutkan bahwa RAPPT telah digunakan dalam pengembangan aplikasi komersial, dimana Prompa dapat diunduh melalui Google Play Store. *Feedback* informal yang didapatkan dari para *developer* yang menggunakannya adalah *generator* kode sangat mudah untuk digunakan dan meningkatkan produktivitas secara signifikan. Selain itu RAPPT juga mendapatkan beberapa *feedback* terkait limitasi. Salah satu limitasi utama yang ditemui adalah RAPPT tidak bisa digunakan setelah kode dilakukan modifikasi. Namun tidak disebutkan secara detail modifikasi kode seperti apakah yang membuat RAPPT tidak dapat digunakan.

Beberapa poin penting yang dapat diambil dari penelitian tersebut adalah sebagai berikut.

- *Domain Specific Language* (DSL) dapat digunakan untuk abstraksi yang tidak terlalu kompleks sebagai input sistem yang mampu membuat *scaffolding* aplikasi android.
- Berdasarkan pengguna sistem RAPPT, sistem *code generator* ini dapat membantu *developer* dalam meningkatkan produktivitas.
- Sistem dengan keluaran *scaffolding code* memberikan fleksibilitas kepada *developer* untuk mengembangkan lebih lanjut aplikasi yang dibuat.

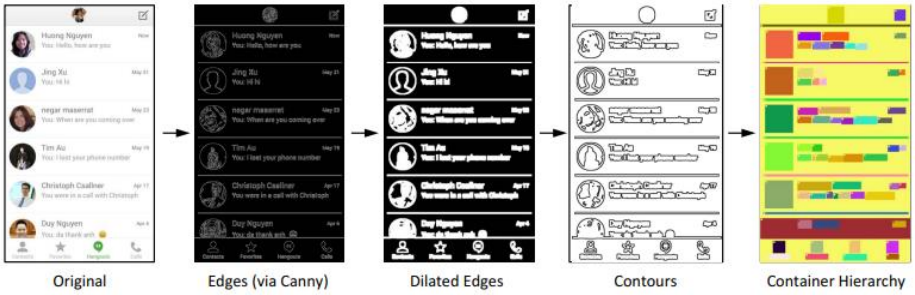
2.1.2 **Reverse Engineering Mobile Application User Interface with REMAUI[3]**

Penelitian ini dilakukan oleh Nguyen, Tuan Anh, dan Csallner, Christop dari *Computer Science and Engineering Department University of Texas* dengan tujuan untuk membuat metode otomasi *reverse engineering user interface* aplikasi *mobile*. REMAUI menggunakan *Optical Character Recognition* (OCR) untuk membuat kode *user interface* yang mendekati konseptual gambar input. Pada Gambar 2.2 dapat dilihat gambaran proses OCR yang digunakan REMAUI memetakan letak tulisan yang kemudian akan digunakan sebagai representasi letak blok kata, baris, dan paragraf.



Gambar 2.2 Gambaran proses tesseract OCR mengidentifikasi letak blok kata, baris, dan paragraf dalam penelitian Reverse Engineering Mobile Application User Interface with REMAUI. [3]

Dalam penerapannya REMAUI mempunyai lima tahapan untuk menghasilkan kode seperti yang tertera pada Gambar 2.3. Tahapan pertama adalah memetakan letak dan mengekstrak kata dan baris menggunakan OCR. Tahapan kedua melakukan ekstraksi *bounding box* setiap elemen UI dengan menggunakan *computer vision*.



Gambar 2.3 Computer vision untuk menentukan *bounding box user interface* dalam REMAUI. [3]

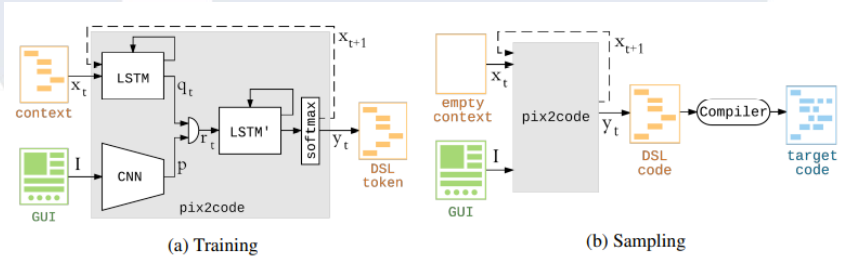
Tahapan ketiga adalah tahap menggabungkan hasil dari proses tahapan OCR dan *computer vision* untuk meningkatkan kualitas rekognisi. Dan tahapan keempat dilakukan identifikasi repetisi *item* dan menjadikannya gabungan sebagai *collections*. Tahapan kelima melakukan *export* konstruksi UI sebagai *resource* aplikasi *mobile* ke dalam *project directory*. Untuk melakukan evaluasi terhadap model yang dibuat, Tuan Anh menggunakan 488 *screenshot* aplikasi *mobile* dan REMAUI mampu menghasilkan *kode user interface* yang mirip dengan gambar input. Namun kode yang dihasilkan hanya dalam satu *platform* kode yaitu android. Dalam kesimpulan penelitiannya Tuan Anh berencana untuk mengintegrasikan REMAUI dengan *tools* yang bisa melakukan generasi kode aplikasi mobile berdasarkan pencarian kode, *high-level models*, ataupun DSL.

Beberapa poin penting yang dapat diambil dalam penelitian tersebut adalah sebagai berikut.

- Penggabungan metode *computer vision* dengan *Optical Character Recognition* (OCR) dapat mengekstrak *features* pada input gambar *user interface* dan mengidentifikasi elemen-elemennya seperti gambar, *text*, *container*, dan baris.

2.1.3 Pix2code: Generating Code From a Graphical User Interface Screenshot[4]

Pada penelitian oleh T. Bertramelli pendekatan untuk menghasilkan kode dengan format input berupa gambar *user interface* adalah dengan arsitektur *encoder-decoder* menggunakan *Convolutional Neural Network* (CNN) dan *Recurrent Neural Network* (RNN). Dijelaskan bahwa Pix2code merumuskan pendekatan ini untuk memanfaatkan kemampuan *machine learning* yang dapat mempelajari variabel laten pada gambar *input*, sehingga mengurangi kompleksitas rekayasa pencarian *features* pada input seperti yang dilakukan pada REMAUI [3].



Gambar 2.4 Arsitektur Pix2code [4].

Arsitektur model Pix2code terdiri dari tiga bagian yaitu visual model menggunakan *end to end* CNN dan *language* model serta *encoder* menggunakan LSTM seperti yang tertera pada Gambar 2.3. Visual model digunakan untuk mengekstrak *feature* pada gambar input dengan susunan dua layer CNN dengan *width* 32, diikuti layer dengan *width* 64 dan layer dengan *width* 128. Kemudian terakhir diberikan layer dengan ukuran 1024 menggunakan aktivasi *rectified linear unit*. Pada *language* model *Domain Specific Language* (DSL) untuk merepresentasikan UI seperti pada Gambar 2.4. Tidak seperti pada REMAUI [3], Pix2code mengabaikan label pada elemen UI dengan tujuan untuk mengurangi kompleksitas pengenalan gambar sehingga mengurangi beban komputasi.

Gambar 2.5 Gambar UI website direpresentasikan dengan DSL token

Keluaran dari visual model *p* berupa representasi *vector* dari gambar input dan keluaran *language* model q_t berupa representasi *intermediary* dari token digabungkan *(concatenated)* menjadi *single feature vector* r_t yang dijadikan input *decoder* menggunakan LSTM untuk mempelajari kaitan antara fitur dari gambar input dan fitur token.

$$p = CNN(I) \quad (1)$$

$$q_t = LSTM(x_t) \quad (2)$$

$$r_t = (q_t, p) \quad (3)$$

$$y_t = softmax(LSTM'(r_t)) \quad (4)$$

$$x_{t+1} = y_t \quad (5)$$

Pada Gambar 2.3 terlihat terdapat perbedaan antara arsitektur untuk training dan sampling, dimana pada arsitektur sampling terdapat *compiler*. *Compiler* tersebut digunakan untuk mengubah DSL token menjadi keluaran kode final yang diinginkan.

Poin penting yang dapat diambil dalam penelitian tersebut adalah sebagai berikut.

- Arsitektur *decoder–encoder* yang umumnya diimplementasikan pada model untuk *image captioning* dapat diimplementasikan untuk membuat model generator kode. Arsitektur ini merupakan salah satu penerapan *sequence to sequence learning.*

- DSL dapat digunakan untuk merepresentasikan elemen yang terdapat pada gambar UI sehingga mengurangi kompleksitas model untuk memprediksi kode.

2.1.4 Pre-trained CNNs as Feature-Extraction Modules for Image Captioning: An Experimental Study[5]

Abdelhadie Al-Malla, Muhammad; Jafar, Assef; Ghneim, Nada pada penelitiannya yang berjudul *Pre-trained CNNs as Feature-Extraction Modules for Image Captioning: An Experimentasl Study* melakukan percobaan untuk menguji 12 arsitektur jenis model VGG, ResNet, Inception, InceptionResNet, DenseNet, dan NASNetLarge untuk mengevaluasi keefektifan model tersebut sebagai fitur ekstraktor. Semua model yang diuji sebelumnya telah dilatih menggunakan dataset ImageNet. Penerapan *pre-trained model* menjadi fitur ekstraktor untuk arsitektur *encoder-decoder* dilakukan dengan menghilangkan layer klasifikasi. Setiap arsitektur model kemudian dilatih dengan dataset MS COCO, Flickr8k, dan Flickr30k untuk *captioning image*. Dari penelitian tersebut ditunjukkan bahwa NASNetLarge mempunyai skor evaluasi terbaik dengan menggunakan dataset MS COCO yang merupakan dataset terbesar. Namun, untuk keseluruhan dataset Xception dan InceptionResNet V2 mempunyai performa yang baik secara konsisten. Adapun metrik yang digunakan untuk mengevaluasi arsitektur tersebut adalah BLEU, METEOR, CIDEr, ROUGE-L, SPICE.

Beberapa poin penting yang dapat diambil dalam penelitian tersebut adalah sebagai berikut.

- Xception dan InceptionResNetV2 menjadi fitur ekstraktor yang paling *robust* dalam *image captioning* yang diujikan dengan tiga dataset berbeda.
- Pengukuran performa model *image captioning* dapat menggunakan metode pengukuran BLEU, METEOR, CIDEr, ROUGE-L, atau SPICE.

2.1.5 A comparison of LSTM and GRU networks for learning symbolic sequences[7]

Cahuantzi, Roberto;Chen, Xinye; dan Guttel, Stefan melakukan penelitian untuk menemukan hubungan antara hyperparameter RNN dan kompleksitas data yang dapat diingat menggunakan dua jenis RNN, yaitu GRU dan LSTM. Kompleksitas diukur melalui panjang input dan jumlah kosa kata yang digunakan. Pada penelitian tersebut digunakan data dengan kompleksitas tinggi yang memiliki panjang input antara 5.000 hingga 10.000 yang terdiri dari 10 hingga 52 kosa kata dan data dengan kompleksitas rendah memiliki panjang input antara 2 hingga 12 yang terdiri dari 2 hingga 6 kosa kata. Penelitian ini menemukan bahwa dua *hyperparameter* yang paling penting adalah *learning rate* dan jumlah layer dalam model. Hasilnya menunjukkan bahwa semakin besar dimensi layer, semakin akurat model tersebut. Kesimpulan pada penilitan ini menunjukkan bahwa GRU menggunakan waktu yang lebih sedikit daripada LSTM pada data dengan kompleksitas rendah. Namun, pada data dengan kompleksitas tinggi, LSTM membutuhkan waktu yang lebih sedikit daripada GRU dan dilihat dari akurasi model, LSTM menghasilkan akurasi yang lebih tinggi di kedua jenis sistem yang diuji daripada GRU.

Beberapa poin penting yang dapat diambil dalam penelitian tersebut adalah sebagai berikut.

- GRU memerlukan waktu yang lebih sedikit dibandingkan LSTM dengan kompleksitas data input rendah. Namun, pada kompleksitas tinggi LSTM membutuhkan waktu yang lebih sedikit dibandingkan GRU.
- LSTM menghasilkan akurasi yang lebih tinggi baik dengan kompleksitas data rendah maupun tinggi.

2.1.6 Rangkuman Penelitian Tedahulu

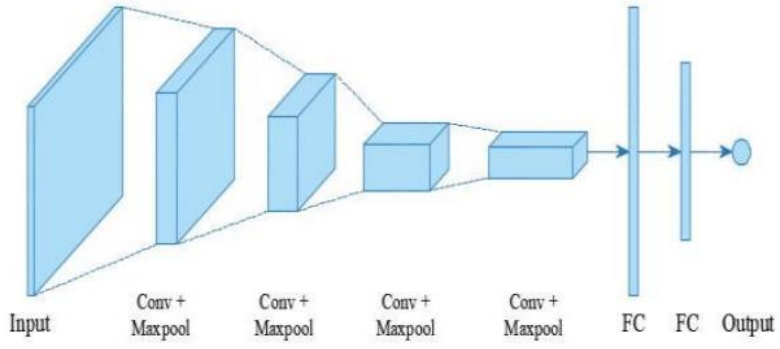
Berdasarkan penelitian-penelitian terdahulu, terdapat beberapa poin yang akan digunakan sebagai acuan untuk implementasi penelitian ini.

1. Susunan arsitektur Pix2code masih menjadi opsi terbaik untuk implementasi kode generator menggunakan *machine learning* sehingga arsitektur yang sama akan digunakan pada penelitian ini. Penggunaan OCR seperti pada arsitektur REMAUI memerlukan pengetahuan khusus untuk menentukan rule *heuristic* agar menghindari *false positive prediction*. Arsitektur Pix2code menggunakan arsitektur *encoder-decocer* dimana visual model menggunakan *end to end* CNN dengan *language* model dan *decoder* LSTM, karenanya uji coba penggunaan kombinasi CNN dan RNN model lain sebagai *encoder-decoder* masih dimungkinkan untuk membuat model yang lebih baik.
2. DSL digunakan untuk merepresentasikan kode sebenarnya. DSL dapat mengurangi kompleksitas kosa kata yang digunakan dan mengurangi beban komputasi. Keluaran *source code* dapat disesuaikan tanpa melakukan pelatihan ulang model dengan cara mengubah membuat *custom* compiler dan mapping DSL yang digunakan untuk pelatihan model.
3. GRU membutuhkan waktu yang lebih sedikit daripada LSTM untuk penggunaan data dengan kompleksitas rendah. Sehingga GRU akan digunakan sebagai *language model* dan *encoder* pada penelitian ini.
4. *Pre-trained* Xception dengan menggunakan dataset ImageNet mempunyai performa yang baik digunakan sebagai fitur ekstraktor pada arsitekur *encoder-decoder* untuk image *captioning*.
5. Xception berpotensi menjadi fitur ekstraktor yang baik namun ukuran model masih tergolong besar. Oleh sebab itu, model yang lebih ringan akan diuji dengan menggunakan model EfficientNetV2B0 dan EfficientNetV2M.

2.2 Tinjauan Teori

2.2.1 Convolutional Neural Network

Convolutional Neural Network (CNN) adalah salah satu jenis jaringan saraf yang telah banyak diterapkan dan terbukti berhasil dalam rekognisi gambar dan video (Krizhevsky et al., 2012; Zeiler & Fergus, 2013, Sermanet et al., 2014; Simonyan & Zisserman, 2014) [13]. Arsitektur CNN terinspirasi dari sistem saraf biologis dimana operasi konvolusi menggabungkan beberapa lapisan pemrosesan yang disebut *neuron* (Hu et al., 2015). Arsitektur CNN secara umum tertera pada Gambar 2.4, keluaran dari setiap tahapan dalam CNN adalah *vector* yang disebut *feature maps*. Dalam kasus input gambar berwarna, setiap *feature maps* adalah array dua diemensi yang berisi *channel* warna dari gambar input [14].



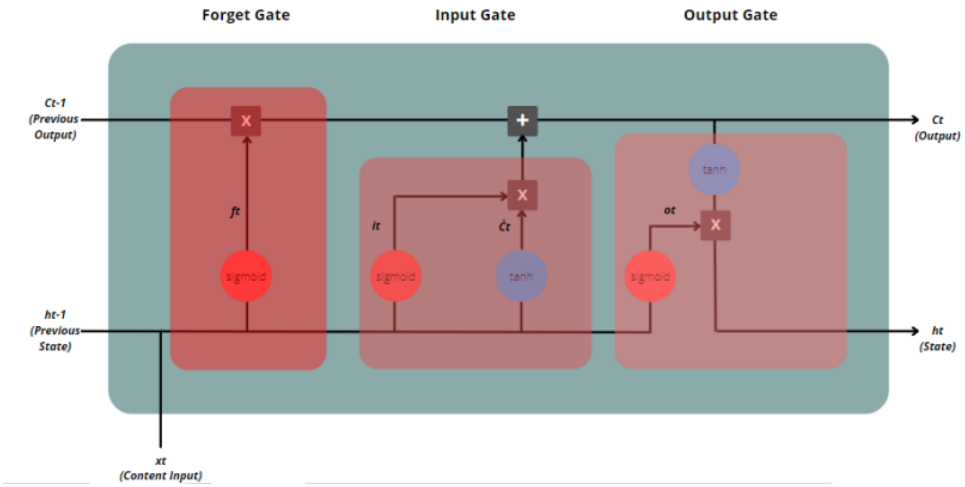
Gambar 2.6 Arstiektur CNN secara umum [14]

2.2.2 Long Short Term Memory

Long Short Term Memory (LSTM) adalah salah satu jenis algoritma yang masih bagian dari *Recurrent Neural Network* (RNN). Perbedaan LSTM dengan algoritma lain adalah kemampuannya dalam melakukan *long-term learning*. LSTM dibuat dengan tujuan untuk memperbaiki kelemahan RNN yaitu *vanishing gradient problem* atau permasalahan luruhnya atau hilangnya efektivitas gradien dalam pembelajaran yang

panjang dan mengakibatkan pemrosesan data tidak efektif [3]. LSTM dirancang dengan struktur *cell* yang lebih kompleks dengan memori dan *gate* yang memungkinkan model memilih untuk menyimpan atau mengabaikan informasi dari *input* sebelumnya. Hal tersebut menjadikan LSTM mampu menangani *long-term dependencies* dalam data yang dimasukkan.

Proses LSTM terbagi menjadi tiga elemen utama, yaitu *cell state*, *hidden state*, dan *input*. Untuk menyimpan informasi dari interval sebelumnya digunakan *cell state*. Representasi prediksi dari *node* LSTM sebelumnya dilakukan oleh *hidden state* dan *input* digunakan sebagai *input* data pada *time step* tersebut. Untuk menjaga agar *cell state* dapat merepresentasikan informasi dari interval sebelumnya dengan tepat, LSTM memanfaatkan *gate*. *Gate* merupakan konsep yang digunakan pada LSTM untuk mengendalikan bagaimana data diproses. Terdapat tiga *gate* yaitu *forget gate*, *input gate*, dan *output gate* seperti yang tertera pada Gambar 2.5.

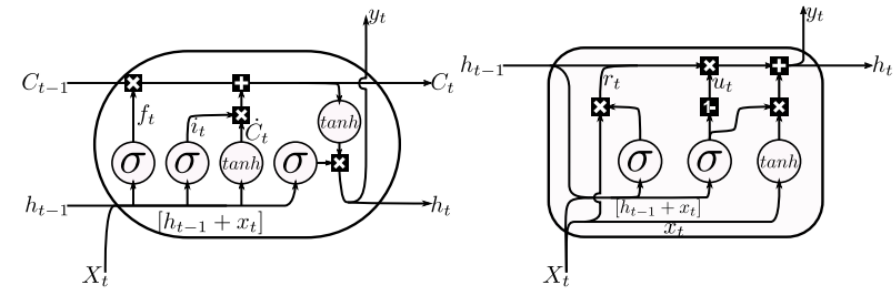


Gambar 2.7 Struktur LSTM [15]

2.2.3 Gated Recurrent Unit

Gated Recurrent Unit (GRU) merupakan salah satu jenis RNN. GRU dibuat dengan tujuan yang sama dengan LSTM, yaitu untuk memperbaiki kelemahan RNN dalam pemrosesan data dalam *long-term learning*.

Perbedaan GRU dengan LSTM adalah GRU menggunakan *training parameter* yang lebih sedikit dan hanya menggunakan dua *gate* [6][7] sehingga secara performa komputasi GRU lebih unggul [6]. Perbedaan struktur antara GRU dan LSTM dapat dilihat pada Gambar 2.7.



Gambar 2.8 Struktur umum LSTM (kanan) dan GRU (kiri)

2.2.4 Xception

Xception merupakan sebuah arsitektur CNN yang dikembangkan oleh Francois Chollet pada tahun 2016. Xception dikembangkan berdasarkan model sebelumnya yaitu Inception. Model Xception menggunakan serangkaian *depthwise separable convolutions* yang memungkinkan model mempunyai banyak layer konvolusi namun dengan kompleksitas komputasi yang relatif rendah [10]. Tidak hanya pada klasifikasi, penggunaan Xception untuk fitur ekstraktor sebagai *encoder* pada arsitektur *encoder-decoder image captioning* juga telah diuji dan menghasilkan performa yang baik menggunakan tiga dataset yaitu MS COCO, Flickr8k, dan Flickr30k [5].

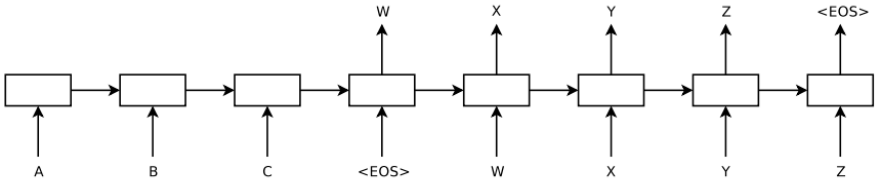
2.2.5 EfficientNetV2B0

EfficientNetV2B0 merupakan salah satu arsitektur CNN yang mempunyai kecepatan *training* dan efisiensi parameter lebih baik dibandingkan model sebelumnya. Efisiensi yang berbeda dari versi sebelumnya antara lain pengguanaan MBConv dan Fused-MBConved yang membuat performa EfficientNetV2 lebih baik pada *mobile* atau akselerator

server [17]. EffisientNetV2 mempunyai model size hanya 29 MB dengan akurasi sekitar 77,1 persen [5].

2.2.6 Sequence to Sequence Learning

Sequence to sequence learning atau biasa dikenal dengan seq2seq adalah teknik *machine learning* yang umum digunakan pada pemrosesan teks [22]. Model seq2seq terdiri dari dua komponen utama, yaitu *encoder* dan *decoder*. *Decoder* berfungsi sebagai *feature extractor* dari *sequence* dengan keluaran yang disebut sebagai vektor konteks. Kemudian *Decoder* melakukan pemrosesan vektor konteks untuk melakukan prediksi berdasarkan vektor konteks dari *decoder*. Selain masukan berupa teks, seq2seq juga dapat digunakan untuk masukan berupa gambar untuk melakukan *image captioning* [21][24]. Secara umum proses pada teknik seq2seq dapat dilihat pada Gambar 2.7.



Gambar 2.9 Proses masukan dan keluaran teknik Seq2Seq[21]

2.2.7 Domain Specific Language

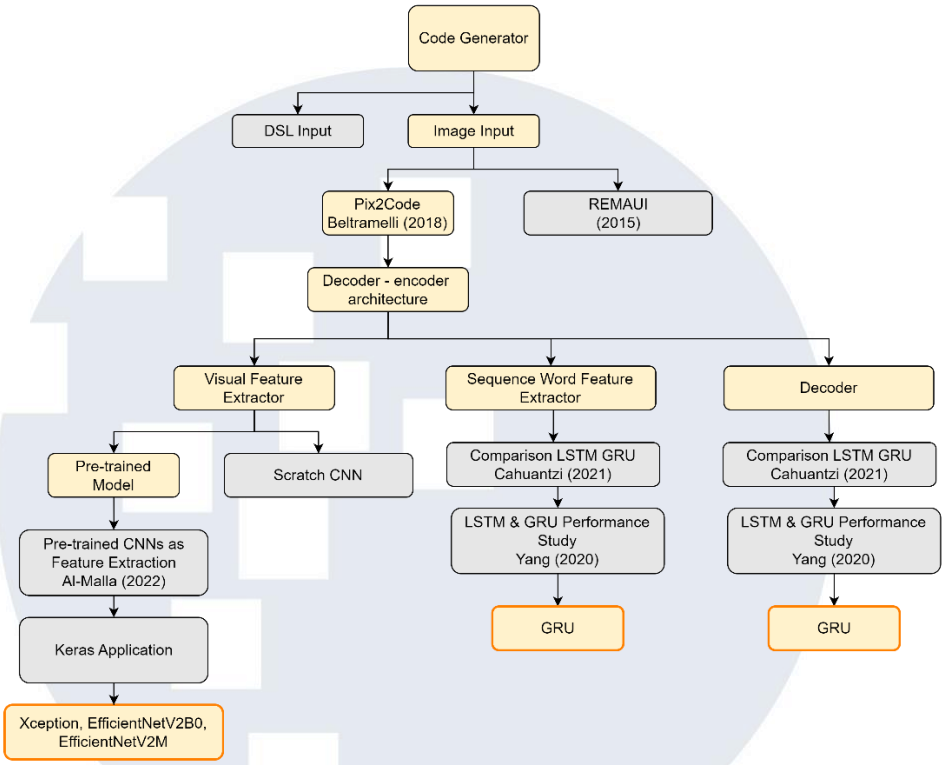
Domain Specific language (DSL) adalah bahasa pemrograman yang dibuat khusus untuk menyelesaikan masalah dalam domain tertentu. DSL dibuat dengan menggunakan notasi yang lebih sederhana dan terstruktur dibandingkan dengan bahasa pemrograman-pemrograman aslinya. DSL biasa digunakan sebagai abstraksi *interface* dengan tujuan mengurangi kompleksitas dalam fungsionalitas kode [18]. Dalam banyak kasus, DSL digunakan untuk memanggil *subroutine library*.

2.2.8 Boilerplate dan NuxtJs

Boilerplate dalam lingkup pengembangan aplikasi merujuk pada unit kode yang dapat digunakan secara berulang-ulang tanpa ada perubahan [9]. Boilerplate merupakan istilah yang digunakan dalam implementasi konsep *reusable programming.* Sebagai contoh boilerplate HTML dapat berisi struktur umum halaman web seperti header, navigasi, footer, dll. yang digunakan dalam seluruh halaman. Boilerplate membantu mempercepat proses pengembangan dan memastikan konsistensi visual. Boilerplate umumnya ditanamkan dalam kerangka pengembangan yang disebut scaffolding atau biasa disebut dengan *starter kit*.

Keluaran sistem yang dibuat pada penelitian ini terdapat pilihan untuk menghasilkan keluaran boilerplate berbasis NuxtJs. NuxtJs merupakan *open source frontend framework* dengan dasar VueJs [19]. Penggunaan *framework* NuxtJs memberikan struktur dan fleksibilitas pengembangan perangkat lunak berbasis web yang lebih baik. Selain itu penggunaan NuxtJs juga cukup populer berdasarkan Stackoverflow Developer Survey 2022 [20].

Berdasarkan penelitian dan studi literatur yang telah dilakukan rangkuman poin yang akan dijadikan acuan dalam penelitian ini tertera pada Gambar 2.9.



Gambar 2.10 State of the art.