BAB 2. TINJAUAN PUSTAKA

2.1 Web Scraping

Web scraping adalah proses otomatisasi pengumpulan, ekstraksi, dan penyimpan data dari halaman web menggunakan perangkat lunak atau skrip pemrograman (Zhao, 2017). Secara umum, *web scraping* bekerja dengan mengirimkan permintaan HTTP ke *server* target dan menganalisis respon HTML yang diterima. Namun, pada platform *e-commerce* modern yang memanfaatkan pemuatan konten secara dinamis (*dynamic client-side rendering*), pendekatan ekstraksi kode sumber HTML mentah (*raw HTML*) sering kali mengalami kendala karena data produk belum sepenuhnya dimuat saat *request* awal dilakukan (Chandra & Tiwari, 2021).

Untuk mengatasi keterbatasan tersebut, teknik *web scraping* berkembang menjadi *DOM-based Scraping* yang berfokus pada pembacaan struktur elemen *Document Object Model* (DOM) di sisi *browser client* setelah seluruh skrip JavaScript selesai dieksekusi. Dalam penelitian ini, *web scraping* diterapkan melalui komponen *Content Script* pada ekstensi peramban untuk mengekstraksi data produk secara *real-time* langsung dari antarmuka visual halaman web 1688.com.

2.2 Document Object Model (DOM)

Document Object Model (DOM) adalah antarmuka pemrograman aplikasi (*Application Programming Interface* / API) standar yang mendefinisikan struktur dokumen HTML dan XML sebagai struktur pohon (*DOM Tree*) berhierarki (W3C, 2020). Setiap elemen, atribut, dan teks di dalam dokumen dipetakan sebagai node/objek yang dapat diakses, dimanipulasi, dan diperbarui secara dinamis menggunakan bahasa pemrograman skrip seperti JavaScript (Flanagan, 2020).

Pada platform *e-commerce* seperti 1688.com yang menggunakan pemuatan konten dinamis berbasis AJAX dan Single Page Application (SPA), struktur HTML mentah yang dikirim oleh server sering kali belum memuat data produk secara utuh (Chandra & Tiwari, 2021). Melalui manipulasi dan pembacaan DOM setelah proses *rendering* selesai pada *browser client*, ekstensi peramban dapat mengekstraksi informasi produk (seperti judul, harga, variasi, dan URL gambar) secara akurat sesuai dengan elemen visual yang sedang ditampilkan kepada pengguna (Varghese & Vijay, 2019).

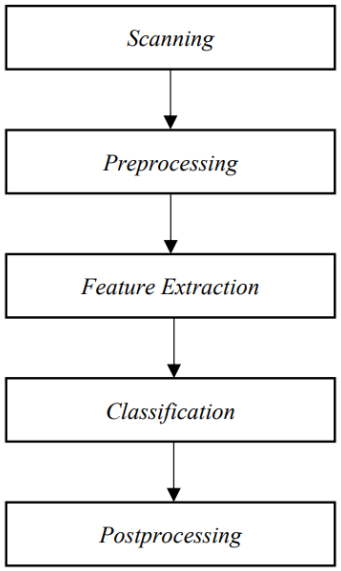
2.3 Optical Character Recognition (OCR)

Teknologi Optical Character Recognition (OCR) berfungsi mengekstrak teks dari gambar, memungkinkan sistem membaca informasi visual secara

otomatis. OCR menjadi elemen penting dalam penelitian ini karena banyak produk 1688 yang menampilkan informasi dalam bentuk teks tertanam di gambar.

OCR modern seperti Tesseract 5.0 atau EasyOCR telah mendukung berbagai bahasa, termasuk Mandarin. Namun, akurasi pengenalan dipengaruhi oleh kualitas gambar, pencahayaan, dan jenis font (Chen et al., 2021).

Akurasi OCR sangatlah krusial dan dipengaruhi oleh kualitas pra-pemrosesan citra, seperti *binarization* dan *noise reduction* (Gabor et al., 2018). Lebih lanjut, karakter Mandarin dikenal memiliki tingkat kesulitan pengenalan yang tinggi. Wang et al. (2020) dalam studi *Hybrid model for Chinese character recognition based on Tesseract-OCR* menunjukkan bahwa akurasi pengenalan karakter Tionghoa dapat ditingkatkan hingga sekitar 12% melalui kombinasi pra-pemrosesan citra yang spesifik dan penggunaan model koreksi pasca-OCR (*post-OCR correction*). Penelitian ini menegaskan pentingnya optimasi tahapan awal (pra-pemrosesan) untuk meningkatkan akurasi ekstraksi teks Mandarin, menjadikannya target utama optimasi dalam penelitian ini.



Gambar 2.1 Proses OCR

2.3.1 EasyOCR Framework

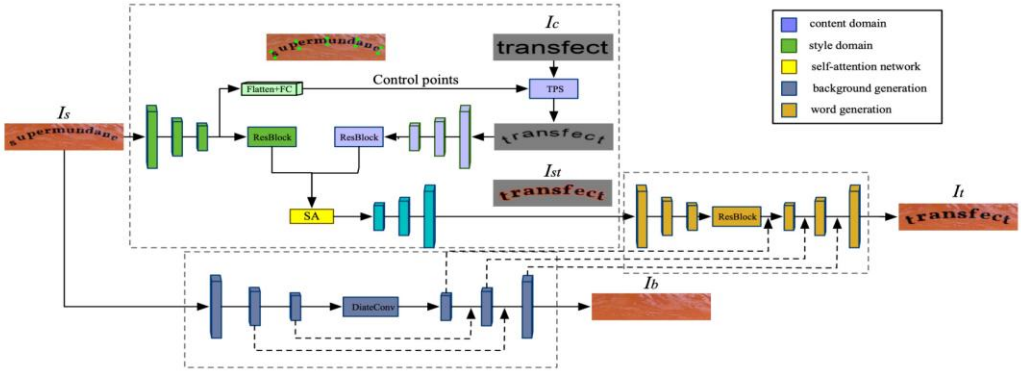
EasyOCR merupakan salah satu pustaka (*library*) OCR berbuka sumber (*open-source*) berbasis Python yang dikembangkan oleh Jaided AI. EasyOCR dibangun di atas kerangka kerja pemelajaran dalam (*Deep Learning*) PyTorch dan dirancang untuk mendukung pengenalan teks multibahasa, termasuk teks berbahasa Mandarin dan Inggris (Jaided AI, 2020).

Dalam arsitekturnya, EasyOCR mengintegrasikan dua model utama, yaitu CRAFT (*Character Region Awareness for Text Detection*) yang berfungsi untuk mendeteksi area/lokasi batas teks (*bounding box*) pada citra, serta CRNN (*Convolutional Recurrent Neural Network*) yang digabungkan dengan *CTC Loss* (*Connectionist Temporal Classification*) untuk mengenali dan memprediksi urutan karakter di dalam area teks tersebut. Keunggulan arsitektur ini menjadikan EasyOCR responsif dan akurat dalam mengektraksi teks dari gambar produk sebelum diteruskan ke tahap penerjemahan dan *image text swapping*.

2.4 Image Text Swapping

Image Text Swapping, yang juga dikenal sebagai *scene text editing*, merupakan teknik pengolahan citra yang bertujuan untuk menghapus teks asli pada suatu gambar dan menggantinya dengan teks baru tanpa merusak struktur visual gambar tersebut. Teknik ini memungkinkan penggantian teks dilakukan dengan tetap mempertahankan konteks visual, seperti warna latar, tekstur, dan tata letak objek di sekitarnya. Menurut Zhang et al. (2021), image text swapping banyak digunakan dalam sistem visual multibahasa untuk meningkatkan keterbacaan informasi pada gambar lintas bahasa.

Setelah proses ekstraksi dan translasi teks (OCR) selesai, tahapan selanjutnya adalah Image Text Swapping atau *Scene Text Editing*. Proses ini adalah inti dari solusi visual multibahasa yang diusulkan. *Image Text Swapping* melibatkan dua langkah kompleks: (1) Inpainting, yaitu penghapusan teks sumber dari gambar, dan (2) Text Rendering, yaitu penyisipan teks terjemahan ke lokasi yang sama dengan tetap mempertahankan atribut visual (*font*, warna, *background*). Yang et al. (2020) dalam paper SwapText: Image Based Texts Transfer in Scenes mengusulkan kerangka yang efektif untuk mentransfer teks antar gambar dengan preservasi tekstur latar belakang.



Gambar 2.2 Arsitektur Model SwapText (Yang et al., 2020)

Penelitian oleh Yang et al. (2020) melalui model *SwapText* menunjukkan bahwa pendekatan berbasis inpainting dan text rendering mampu menghasilkan penggantian teks yang lebih natural dengan preservasi tekstur latar belakang.

Dalam konteks penelitian ini, proses *image text swapping* akan disesuaikan menggunakan pendekatan OCR (Optical Character Recognition) untuk mengekstrak teks Mandarin dari gambar produk, kemudian menerjemahkannya menggunakan model NLP multibahasa seperti MarianNMT atau Gemini API, dan akhirnya mengganti teks lama dengan teks hasil translasi menggunakan kombinasi OpenCV dan Pillow.

2.5 Firefly Algorithm

2.5.1 Pengertian Firefly Algorithm

Firefly Algorithm (FA) merupakan salah satu algoritma optimasi berbasis populasi (*population-based metaheuristic*) yang diperkenalkan oleh Xin-She Yang pada tahun 2008. Algoritma ini terinspirasi dari perilaku alami kunang-kunang (*fireflies*) yang menggunakan intensitas cahaya sebagai mekanisme komunikasi untuk menarik individu lain. Dalam Firefly Algorithm, tingkat kecerahan (*brightness*) merepresentasikan kualitas solusi (*fitness*), sehingga individu dengan tingkat kecerahan yang lebih rendah akan bergerak menuju individu dengan tingkat kecerahan yang lebih tinggi *(Yang, 2008).*

Firefly Algorithm termasuk ke dalam kelompok algoritma *swarm intelligence* yang memanfaatkan interaksi antarindividu dalam suatu populasi untuk memperoleh solusi optimal. Algoritma ini memiliki kemampuan yang baik dalam menyeimbangkan proses eksplorasi (*exploration*) dan eksploitasi (*exploitation*), sehingga banyak diterapkan pada berbagai permasalahan optimasi, seperti optimasi parameter, penjadwalan (*scheduling*), alokasi sumber daya, serta optimasi jaringan *(Yang, 2009).*

Seiring perkembangannya, Firefly Algorithm terus mengalami pengembangan untuk menyelesaikan permasalahan optimasi yang lebih kompleks. Salah satu penerapannya adalah optimasi *workflow scheduling* pada lingkungan *hybrid cloud-edge* melalui modifikasi mekanisme Firefly untuk meningkatkan efisiensi penggunaan sumber daya, menurunkan *makespan*, serta meningkatkan *resource utilization*. Hasil penelitian tersebut menunjukkan bahwa Firefly Algorithm tetap relevan dan mampu memberikan performa yang baik pada permasalahan penjadwalan modern *(Alsadie & Alsulami, 2024).*

2.5.2 Pengertian Kerja Firefly Algorithm

Firefly Algorithm bekerja berdasarkan tiga aturan dasar yang mengadopsi perilaku alami kunang-kunang. Pertama, seluruh kunang-kunang dianggap tidak memiliki perbedaan jenis kelamin (*unisex*), sehingga setiap individu dapat saling menarik tanpa memperhatikan jenisnya. Kedua, tingkat ketertarikan (*attractiveness*) antar kunang-kunang berbanding lurus dengan intensitas cahaya (*brightness*) yang dipancarkan, sehingga kunang-kunang dengan tingkat kecerahan yang lebih rendah akan bergerak menuju kunang-kunang yang lebih terang. Ketiga, tingkat kecerahan setiap kunang-kunang ditentukan oleh nilai fungsi objektif (*fitness function*), sehingga solusi dengan nilai *fitness* yang lebih baik akan memiliki tingkat kecerahan yang lebih tinggi *(Yang, 2009).*

Pada Firefly Algorithm, jarak antara dua kunang-kunang dihitung menggunakan jarak Euclidean yang dinyatakan sebagai berikut *(Yang, 2009).*

$$r_{ij} = \sqrt{\sum_{k=1}^{d}(x_{i,k} - x_{j,k})^2} \qquad (2.1)$$

Keterangan:

- r_{ij}= jarak antara kunang-kunang ke-*i* dan ke-*j*
- $x_{i,k}$ = posisi kunang-kunang ke-*i* pada dimensi ke-*k*
- $x_{j,k}$ = posisi kunang-kunang ke-*j* pada dimensi ke-*k*
- d = jumlah dimensi pencarian

Semakin dekat jarak antar kunang-kunang, semakin besar pula daya tarik yang dihasilkan. Sebaliknya, daya tarik akan berkurang secara eksponensial seiring bertambahnya jarak antar individu. Hubungan tersebut dinyatakan melalui persamaan *attractiveness* berikut *(Yang, 2009).*

$$\beta(r) = \beta 0 e^{\gamma r_{ij}^2} \qquad (2.2)$$

Keterangan:

- $\beta(r)$ = tingkat ketertarikan pada jarak *r*
- $\beta 0$ = tingkat ketertarikan maksimum saat *r* = 0
- γ = koefisien penyerapan cahaya (*light absorption coefficient*)
- r_{ij} = jarak antar kunang-kunang

Pergerakan kunang-kunang menuju solusi yang lebih baik dilakukan dengan memperhatikan tingkat ketertarikan serta faktor acak (*randomization*) untuk

menjaga keseimbangan antara proses eksplorasi dan eksploitasi. Persamaan perpindahan posisi kunang-kunang dirumuskan sebagai berikut *(Yang, 2009).*

$$x_i = x_i + \beta 0 e^{\gamma r_{ij}^2}(x_j - x_i) + \alpha(rand - 0.50) \qquad (2.3)$$

Keterangan:

- x_i = posisi kunang-kunang ke-*i*
- x_j = posisi kunang-kunang yang memiliki tingkat kecerahan lebih tinggi
- $\beta 0$ = tingkat ketertarikan maksimum
- γ = koefisien penyerapan cahaya
- α = parameter randomisasi
- rand = bilangan acak dengan rentang 0 hingga 1

Setelah posisi kunang-kunang diperbarui, setiap solusi dievaluasi menggunakan fungsi objektif (*fitness function*) untuk menentukan tingkat kualitas solusi yang diperoleh. Nilai *fitness* digunakan sebagai dasar dalam menentukan tingkat kecerahan (*brightness*) setiap kunang-kunang sehingga solusi dengan nilai *fitness* yang lebih baik akan memiliki tingkat kecerahan yang lebih tinggi. Secara umum, fungsi *fitness* pada Firefly Algorithm dinyatakan sebagai berikut (Yang, 2009).

$$Fitness = f(x) \qquad (2.4)$$

Keterangan:
$Fitness$ = nilai kualitas solusi yang diperoleh.
$f(x)$ = fungsi objektif yang digunakan untuk mengevaluasi suatu solusi, dengan bentuk yang disesuaikan terhadap permasalahan optimasi yang diselesaikan.

Bentuk fungsi *fitness* pada Firefly Algorithm tidak bersifat tetap, melainkan disesuaikan dengan karakteristik permasalahan yang dioptimasi. Oleh karena itu, setiap penelitian dapat mendefinisikan fungsi *fitness* yang berbeda sesuai tujuan optimasi. Pada penelitian ini, fungsi *fitness* dirancang berdasarkan indikator throughput, parameter match, resource efficiency, dan retry rate. Perancangan fungsi *fitness* tersebut dijelaskan lebih lanjut pada Subbab 3.3.3.

Berdasarkan persamaan tersebut, proses optimasi pada Firefly Algorithm dilakukan secara iteratif. Setiap kunang-kunang terlebih dahulu dievaluasi menggunakan fungsi objektif untuk memperoleh tingkat kecerahan (brightness). Selanjutnya, jarak antar kunang-kunang dihitung menggunakan jarak Euclidean,

kemudian nilai daya tarik (attractiveness) ditentukan berdasarkan jarak tersebut. Kunang-kunang dengan tingkat kecerahan yang lebih rendah akan bergerak menuju kunang-kunang yang lebih terang menggunakan persamaan perpindahan. Proses ini dilakukan secara berulang hingga diperoleh solusi dengan nilai fungsi objektif terbaik.

Prinsip-prinsip tersebut memungkinkan Firefly Algorithm digunakan pada berbagai permasalahan optimasi, termasuk optimasi penjadwalan (*scheduling*) dan alokasi sumber daya. Kemampuan algoritma dalam menyeimbangkan eksplorasi dan eksploitasi menjadikannya sesuai untuk diterapkan pada sistem yang memerlukan penyesuaian konfigurasi secara adaptif terhadap perubahan kondisi lingkungan *(Alsadie & Alsulami, 2024).*

Firefly Algorithm dipilih pada penelitian ini karena memiliki kemampuan yang baik dalam menyelesaikan permasalahan optimasi yang melibatkan banyak kombinasi parameter tanpa memerlukan model matematis yang kompleks. Pada sistem yang dikembangkan, konfigurasi *Pipeline Scheduler* terdiri atas beberapa parameter, yaitu jumlah *Download Worker*, *OCR Worker*, *Translation Worker*, dan *Image Text Swapping Worker*. Kombinasi dari parameter-parameter tersebut membentuk ruang pencarian yang cukup besar sehingga diperlukan algoritma optimasi untuk memperoleh konfigurasi yang paling sesuai.

Dibandingkan dengan pendekatan *rule-based* yang menggunakan nilai parameter tetap, Firefly Algorithm mampu melakukan pencarian solusi secara adaptif berdasarkan kondisi aktual pipeline. Algoritma mengevaluasi setiap kandidat konfigurasi menggunakan fungsi *fitness* yang mempertimbangkan indikator performa seperti *throughput*, *parameter matching*, *resource efficiency*, dan *retry rate*. Dengan mekanisme tersebut, konfigurasi worker yang dihasilkan dapat menyesuaikan perubahan beban kerja sehingga distribusi proses pada setiap tahapan pipeline menjadi lebih seimbang.

Selain itu, Firefly Algorithm memiliki mekanisme keseimbangan antara proses eksplorasi (*exploration*) dan eksploitasi (*exploitation*) sehingga mampu mencari solusi yang baik tanpa terjebak pada solusi lokal. Karakteristik tersebut sesuai dengan kebutuhan penelitian ini yang memerlukan optimasi konfigurasi *Pipeline Scheduler* secara adaptif untuk meningkatkan efisiensi proses scraping dan *image text swapping* pada platform 1688.

2.5.3 Implementasi Firefly Algorithm pada Penelitian

Pada penelitian ini, Firefly Algorithm diimplementasikan sebagai metode optimasi untuk menentukan konfigurasi *pipeline scheduler* pada proses scraping dan *image text swapping*. Algoritma ini digunakan untuk mengoptimalkan jumlah

proses yang dapat berjalan secara bersamaan (*concurrency limit*) pada setiap tahapan pemrosesan data sehingga penggunaan sumber daya sistem menjadi lebih efisien.

Setiap individu (*firefly*) merepresentasikan satu konfigurasi *concurrency limit* yang terdiri atas lima variabel keputusan (*decision variables*), yaitu batas jumlah proses ekstraksi URL gambar dari *Document Object Model* (DOM), pengunduhan gambar, *Optical Character Recognition* (OCR), penerjemahan menggunakan Google Translate API, dan *image text swapping*. Setiap konfigurasi memiliki kombinasi nilai yang berbeda dan akan dievaluasi untuk memperoleh konfigurasi yang memberikan performa terbaik bagi sistem.

Kualitas setiap konfigurasi diukur menggunakan fungsi *fitness* yang mempertimbangkan beberapa indikator performa *pipeline*, yaitu *throughput*, *makespan*, *pipeline balance*, dan *retry count*. Konfigurasi dengan nilai *fitness* terbaik akan dipilih sebagai konfigurasi *scheduler* yang digunakan dalam proses pemrosesan data.

Proses optimasi dilakukan secara adaptif melalui mekanisme *Performance Monitor* yang memantau kondisi *pipeline* selama sistem berjalan. Ketika terdeteksi perubahan performa, seperti penurunan *throughput* atau ketidakseimbangan beban kerja antar tahapan, Firefly Algorithm akan dijalankan kembali untuk memperoleh konfigurasi *scheduler* yang lebih sesuai. Konfigurasi terbaik kemudian diterapkan secara dinamis tanpa menghentikan proses yang sedang berlangsung sehingga sistem mampu menyesuaikan diri terhadap perubahan beban kerja.

Melalui penerapan tersebut, Firefly Algorithm diharapkan mampu meningkatkan efisiensi proses scraping dan *image text swapping* dengan mengoptimalkan pemanfaatan sumber daya pada setiap tahapan *pipeline*. Selain mempercepat proses pemrosesan data, mekanisme ini juga membantu menjaga keseimbangan beban kerja antar proses sehingga kinerja sistem menjadi lebih stabil.

Dalam penelitian ini, Firefly Algorithm tidak digunakan untuk proses pengolahan citra maupun peningkatan akurasi OCR, melainkan diterapkan sebagai algoritma optimasi metaheuristik pada komponen *Pipeline Scheduler*. Algoritma ini bertugas mengoptimalkan urutan eksekusi tugas (*task scheduling*) dan alokasi daya pada *backend server* agar waktu pemrosesan total (*makespan*) dari rangkaian proses scraping, OCR, translasi, hingga *image text swapping* dapat berjalan secara efisien.

2.6 Ekstensi Chrome (Manifest V3)

Ekstensi Chrome (*Chrome Extension*) merupakan perangkat lunak berbasis peramban (*browser*) yang dikembangkan untuk menambahkan fungsi tertentu pada

Google Chrome. Ekstensi ini memungkinkan pengembang memperluas kemampuan *browser* melalui integrasi dengan halaman web, antarmuka pengguna, maupun berbagai *Application Programming Interface* (API) yang disediakan oleh Chrome. Menurut Google Chrome for Developers (2024), arsitektur terbaru Manifest V3 (MV3) dikembangkan dengan fokus pada peningkatan keamanan, privasi, serta efisiensi penggunaan sumber daya dibandingkan Manifest V2.

Pemilihan Ekstensi Chrome berbasis Manifest V3 sebagai komponen *client-side* dalam penelitian ini didasari oleh beberapa alasan teknis utama:

1. Akses Langsung ke *Rendered DOM* secara *Real-Time*

   Situs *e-commerce* 1688.com menerapkan proteksi pemuatan dinamis berbasis JavaScript. Dengan berjalan sebagai ekstensi peramban, *Content Script* memiliki akses *native* langsung ke *Document Object Model* (DOM) yang telah selesai dirender oleh *browser* pengguna, sehingga mengeliminasi kendala kegagalan ekstraksi data akibat *dynamic rendering* tanpa perlu memicu sistem perlindungan *anti-bot/anti-scraping* dari server target (Chandra & Tiwari, 2021; Varghese & Vijay, 2019).

2. Efisiensi Penggunaan Sumber Daya Sistem

   Berbeda dengan teknik *scraping* tradisional berbasis *server-side headless browser* (seperti Selenium atau Puppeteer) yang membutuhkan konsumsi memori (RAM) dan CPU sangat tinggi untuk menjalankan *instance browser* di *server*, penggunaan Ekstensi Chrome memanfaatkan *browser* yang sedang digunakan oleh pengguna (*client-side execution*). Hal ini secara signifikan mengurangi beban komputasi dan *overhead* pada *server backend* (Klosowski et al., 2022).

3. Integrasi Antarmuka yang Interaktif dan *User-Friendly*

   Ekstensi Chrome memungkinkan integrasi tombol dan panel kontrol secara langsung (*DOM injection*) pada halaman produk 1688.com, sehingga pengguna dapat memicu proses *scraping*, OCR, hingga penerjemahan secara presisi hanya dengan satu klik tanpa harus berpindah aplikasi (Lam, 2021).

Manifest V3 memperkenalkan perubahan mendasar pada arsitektur ekstensi Chrome dengan menggantikan mekanisme *background page* menjadi Service Worker yang bersifat *event-driven*. Pendekatan tersebut memungkinkan ekstensi hanya dijalankan ketika diperlukan sehingga mampu mengurangi penggunaan memori dan meningkatkan efisiensi pemrosesan (*Google Chrome for Developers*, 2024). Selain itu, Manifest V3 juga memperketat mekanisme perizinan

(*permissions*) serta membatasi eksekusi kode dinamis sebagai upaya meningkatkan keamanan ekstensi.

Secara umum, arsitektur Chrome Extension pada Manifest V3 terdiri atas beberapa komponen utama, yaitu Content Script, Service Worker, dan Extension User Interface. Content Script merupakan skrip JavaScript yang dijalankan pada halaman web sehingga memiliki akses terhadap Document Object Model (DOM) untuk membaca maupun memodifikasi elemen halaman sesuai izin yang diberikan. Sementara itu, Service Worker bertugas menangani proses di latar belakang (*background processing*), seperti komunikasi dengan layanan eksternal, pengelolaan penyimpanan data, serta pengendalian logika aplikasi. Adapun Extension User Interface menyediakan antarmuka interaksi antara pengguna dengan ekstensi melalui *popup*, ikon toolbar, maupun halaman pengaturan (*options page*) (Lam, 2021).

Komunikasi antar komponen pada Manifest V3 dilakukan menggunakan mekanisme Message Passing, yaitu pertukaran pesan antara *Content Script, Service Worker*, maupun komponen antarmuka pengguna. Mekanisme ini memungkinkan setiap komponen menjalankan tugas sesuai fungsinya tanpa saling bergantung secara langsung sehingga meningkatkan modularitas dan efisiensi pengembangan sistem (*Google Chrome for Developers*, 2024).

Manifest V3 merupakan standar terbaru dalam pengembangan Chrome Extension yang menggantikan Manifest V2 dengan peningkatan pada aspek keamanan, privasi, serta efisiensi pengelolaan proses latar belakang (background processing). Selain itu, Manifest V3 tetap menyediakan mekanisme Content Script, Service Worker, dan Message Passing yang menjadi komponen utama dalam pengembangan ekstensi modern (Google Chrome for Developers, 2024).

2.7 Pipeline Scheduler

Pipeline Scheduler merupakan komponen pengatur alur kerja (*workflow execution manager*) yang bertugas mengelola, menjadwalkan, serta mengoordinasikan urutan eksekusi tugas-tugas (*tasks*) dalam sebuah sistem pemrosesan data bertahap (*multi-stage processing pipeline*) (Deelman et al., 2019). Dalam arsitektur pemrosesan data terintegrasi seperti rantai proses yang melibatkan ekstraksi DOM, pengunduhan citra, Optical Character Recognition (OCR), ekstraksi terjemahan, dan rekonstruksi citra (*image text swapping*) *scheduler* mengontrol alokasi sumber daya, tingkat eksekusi sejajar (*concurrency level*), antrean tugas (*task queuing*), serta penanganan kueri berulang saat terjadi kegagalan (*retry mechanism*) (Silberschatz et al., 2018).

Penjadwalan *pipeline* yang efisien dirancang untuk meminimalkan waktu penyelesaian total (*makespan*) dan meningkatkan jumlah tugas yang terselesaikan per satuan waktu (*throughput*). Dalam lingkungan pemrosesan asinkron yang melibatkan panggilan API pihak ketiga (seperti Google Translate API dan LLM), *pipeline scheduler* mencegah terjadinya kemacetan pemrosesan (*bottlenecking*) dan menjaga stabilitas sistem agar tidak melampaui batas laju permintaan (*rate limit*) maupun menguras memori (*resource exhaustion*) (Klosowski et al., 2022).

2.8 Flask

Flask merupakan sebuah *micro-framework* berbasis bahasa pemrograman Python yang dikembangkan oleh Armin Ronacher di bawah bendera Pocoo (Grinberg, 2018). Didesain dengan prinsip ketersediaan komponen inti yang minimalis (*lightweight*), Flask tidak mewajibkan pustaka atau alat (*tools*) tertentu dan tidak menyediakan lapisan abstraksi database secara bawaan. Hal ini memberikan fleksibilitas tinggi bagi pengembang untuk memilih dan mengintegrasikan modul ekstensi sesuai kebutuhan spesifik aplikasi (Kroch, 2021).

Flask mengandalkan dua pustaka utama, yaitu WSGI (*Web Server Gateway Interface*) toolkit bernama Werkzeug untuk menangani aspek komunikasi jaringan dan routing HTTP, serta Jinja2 sebagai mesin pembuat templat (*template engine*) (Ronacher, 2020). Karena keunggulannya yang memiliki jejak memori (*overhead*) minim, Flask sangat cocok digunakan untuk membangun layanan antarmuka pemrograman aplikasi (*Application Programming Interface* / API) berbasis RESTful, khususnya pada sistem berbasis Python (Grinberg, 2018).

Dalam arsitektur penelitian ini, Flask difungsikan sebagai penyedia *backend server* yang menerima permintaan (*request*) dari ekstensi peramban (*client*). Flask bertindak sebagai pengatur alur kerja yang menjembatani pertukaran data JSON dan mengoordinasikan eksekusi modul-modul pemrosesan berat di sisi *server*, seperti pemrosesan OCR, integrasi API penerjemah, algoritma *Image Text Swapping*, komunikasi dengan LLM Gemini, serta modul optimasi *Pipeline Scheduler* berbasis *Firefly Algorithm* (Kroch, 2021).

2.9 Platform 1688

1688.com merupakan platform perdagangan elektronik (*e-commerce*) berbasis Business-to-Business (B2B) yang dimiliki oleh Alibaba Group. Platform ini menghubungkan produsen dan pemasok di Tiongkok dengan pelaku usaha, distributor, maupun importir yang membutuhkan produk dalam jumlah besar. Dengan banyaknya variasi produk serta harga yang kompetitif, 1688.com menjadi salah satu platform yang banyak dimanfaatkan sebagai sumber pengadaan produk oleh pelaku usaha impor dari berbagai negara.

Sebagai ilustrasi, struktur halaman produk pada platform 1688.com ditunjukkan pada Gambar 2.3.

Gambar 2.3 Struktur Web 1688

Sebagian besar antarmuka, informasi produk, serta komunikasi pada platform 1688.com disajikan dalam bahasa Mandarin sehingga menjadi salah satu kendala bagi pengguna internasional yang tidak memahami bahasa tersebut (Fasdana, 2025). Selain hambatan bahasa, informasi produk pada 1688.com juga memiliki karakteristik yang berbeda dibandingkan platform *e-commerce* lainnya. Sebagian informasi penting, seperti spesifikasi produk, ukuran, maupun informasi promosi, sering kali disajikan dalam bentuk teks yang menyatu pada gambar produk (*in-image text*). Di sisi lain, informasi produk pada halaman web dimuat secara dinamis menggunakan JavaScript sehingga proses pengambilan data memerlukan pendekatan yang mampu membaca struktur halaman setelah proses *rendering* selesai.

Karakteristik tersebut menjadikan 1688.com sebagai platform yang memiliki tantangan tersendiri dalam proses pengambilan data maupun pemahaman informasi produk. Oleh karena itu, berbagai pendekatan seperti *web scraping*, *Optical Character Recognition (OCR)*, dan penerjemahan otomatis banyak dimanfaatkan untuk membantu pengguna memperoleh informasi produk secara lebih mudah. Pada penelitian ini, karakteristik tersebut menjadi dasar dalam pengembangan sistem untuk mendukung penyusunan katalog produk multibahasa.

2.10 Unified Modeling Language (UML)

Menurut Booch, Rumbaugh, dan Jacobson (2023), Unified Modeling Language (UML) adalah bahasa pemodelan standar yang digunakan untuk merancang, memvisualisasikan, dan mendokumentasikan sistem perangkat lunak berbasis *object-oriented*. UML berfungsi sebagai alat bantu untuk memahami

struktur sistem secara konseptual sebelum tahap implementasi dilakukan. Dengan UML, pengembang dapat mengidentifikasi interaksi antar komponen, alur data, serta hubungan antar proses yang terjadi di dalam sistem.

UML banyak digunakan dalam pengembangan perangkat lunak modern karena mampu memberikan gambaran sistem yang lebih terstruktur dan mudah dipahami.
Seperti dijelaskan oleh López & García (2021), UML membantu dalam mengurangi ambiguitas pada tahap analisis kebutuhan dan meningkatkan kolaborasi antar pengembang, analis, dan pengguna akhir.

Dalam penelitian ini, UML digunakan untuk menggambarkan interaksi antara pengguna (importir) dan sistem yang dikembangkan, yaitu sistem ekstensi Chrome Firefly Algorithm untuk optimasi proses scraping dan image text swapping pada penyusunan katalog produk multibahasa platform 1688.

2.10.1 Activity Diagram

Activity Diagram menggambarkan aliran kerja (*workflow*) atau urutan aktivitas dalam sistem, mulai dari titik awal (*initial state*), mengeksekusi serangkaian alur keputusan (*decision nodes*), hingga mencapai kondisi akhir (*final state*) (Dumas et al., 2018). Dalam penelitian ini, *activity diagram* difungsikan untuk memetakan alur eksekusi data secara mendalam, seperti alur pembacaan DOM oleh *Content Script* Ekstensi Chrome, pengiriman *payload* HTTP ke *Flask REST API*, mekanisme antrean tugas pada *Pipeline Scheduler*, hingga rekonstruksi citra produk.

2.10.2 Class Diagram

Class Diagram memodelkan struktur statis sistem dengan menampilkan kelas-kelas (*classes*), atribut, metode/fungsi (*methods*), serta hubungan asosiasi maupun himpunan (*inheritance/aggregation*) antarkelas dalam perangkat lunak (Pressman & Maxim, 2020). Diagram ini digunakan untuk merancang struktur data *backend* Python (Flask), modul pengelola *EasyOCR*, struktur entitas data produk 1688, serta variabel representasi posisi dan atribut dalam *Firefly Algorithm*.

2.10.3 Sequence Diagram

Sequence Diagram memodelkan interaksi antar objek berdasarkan urutan waktu (*time-ordered interaction*) (Sommerville, 2016). Diagram ini memberikan visualisasi yang jelas mengenai bagaimana komponen-komponen terpisah seperti *Browser UI/Content Script*, *Service Worker (Background)*, *Flask REST API Server*, *Pipeline Scheduler*, dan API eksternal (Google Translate/Gemini) saling bertukar

pesan (*messages/HTTP requests*) secara asinkron dalam satu siklus pemrosesan lengkap.

2.11 Large Language Model (LLM) Gemini

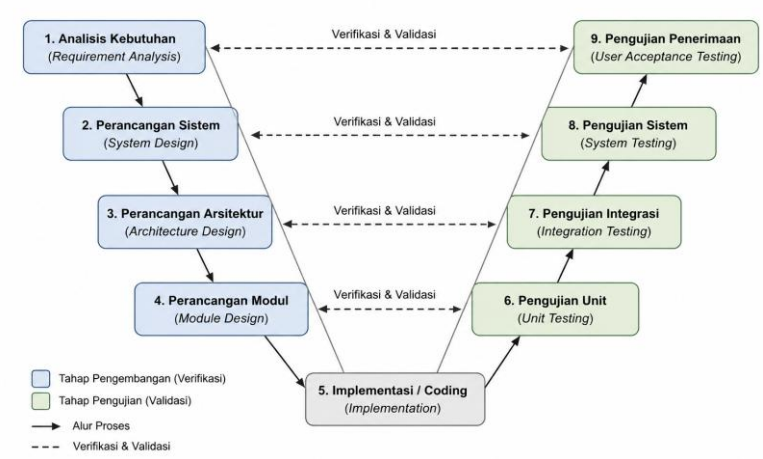
Gemini merupakan model bahasa besar (*Large Language Model* / LLM) yang dikembangkan oleh Google DeepMind. Model ini dirancang secara multimodal sehingga mampu memahami berbagai jenis masukan, seperti teks, gambar, audio, video, dan kode dalam satu arsitektur terpadu (Google DeepMind, 2023).

Salah satu kemampuan utama Gemini adalah menghasilkan teks yang koheren, memahami konteks semantik, melakukan penalaran terhadap informasi yang diberikan, serta menyusun ringkasan maupun deskripsi berdasarkan instruksi pengguna. Kemampuan tersebut menjadikan Gemini banyak dimanfaatkan pada berbagai aplikasi berbasis kecerdasan buatan generatif yang memerlukan penyusunan informasi secara otomatis.

Dalam penelitian ini, Gemini dimanfaatkan untuk menghasilkan ringkasan deskripsi produk berbahasa Indonesia berdasarkan informasi hasil ekstraksi OCR dan metadata produk yang diperoleh melalui proses scraping. Dengan pendekatan tersebut, sistem mampu menyajikan deskripsi produk yang lebih ringkas, mudah dipahami, dan sesuai dengan konteks produk sehingga dapat membantu pengguna dalam melakukan analisis sebelum proses pembelian.

2.12 V-Model (Verification and Validation Model)

Verification and Validation Model (V-Model) merupakan salah satu model *Software Development Life Cycle (SDLC)* yang menekankan hubungan antara setiap tahapan pengembangan perangkat lunak dengan tahapan pengujian yang sesuai. Berbeda dengan model Waterfall yang memisahkan proses pengembangan dan pengujian, V-Model mengintegrasikan aktivitas verifikasi dan validasi sejak awal proses pengembangan sehingga setiap tahapan dapat diuji secara sistematis. Pendekatan ini bertujuan untuk memastikan bahwa perangkat lunak dibangun sesuai dengan kebutuhan pengguna serta memenuhi spesifikasi yang telah ditetapkan. (OpenStax, 2024).



Gambar 2.4 Verification and Validation Model (V-Model)

Sumber: Diadaptasi dari OpenStax (2024).

Gambar 2.4 menunjukkan tahapan V-Model yang terdiri atas dua sisi utama. Sisi kiri menggambarkan tahapan pengembangan perangkat lunak yang dimulai dari analisis kebutuhan, perancangan sistem, perancangan arsitektur, hingga perancangan modul. Setelah seluruh rancangan selesai, proses dilanjutkan pada tahap implementasi *(implementation/coding)* sebagai titik penghubung kedua sisi model.

Setelah proses implementasi selesai, tahapan pada sisi kanan dilakukan sebagai proses validasi terhadap setiap hasil pengembangan. Pengujian dimulai dari *unit testing*, dilanjutkan dengan *integration testing*, *system testing*, hingga *user acceptance testing*. Setiap tahapan pengujian memiliki keterkaitan langsung dengan tahapan perancangan pada sisi kiri sehingga kesalahan yang ditemukan dapat ditelusuri kembali ke tahap pengembangan yang bersesuaian.

Pada penelitian ini, V-Model dipilih sebagai metode pengembangan perangkat lunak karena mampu menghubungkan setiap tahapan perancangan dengan proses pengujian yang sesuai. Pendekatan ini memudahkan proses verifikasi terhadap setiap fitur yang dikembangkan, seperti proses scraping halaman produk, ekstraksi teks menggunakan OCR, penerjemahan menggunakan Google Translate API, image text swapping, penyusunan deskripsi produk menggunakan Gemini, serta optimasi pipeline menggunakan Firefly Algorithm. Dengan demikian, setiap komponen sistem dapat diuji secara bertahap sehingga mengurangi kemungkinan terjadinya kesalahan pada tahap implementasi akhir.

2.13 API Google Translate

Google Translate API merupakan layanan penerjemahan mesin (*Machine Translation*) yang disediakan oleh Google Cloud. API ini memungkinkan aplikasi

menerjemahkan teks secara otomatis ke berbagai bahasa melalui antarmuka berbasis REST tanpa memerlukan pembangunan model penerjemahan secara mandiri. Google Translate API mendukung lebih dari 100 bahasa dan memanfaatkan teknologi *Neural Machine Translation (NMT)* untuk menghasilkan terjemahan yang lebih alami dan mempertahankan konteks kalimat.

Dalam penelitian ini, Google Translate API digunakan untuk menerjemahkan teks berbahasa Mandarin yang diperoleh dari proses OCR menjadi bahasa Indonesia. Hasil terjemahan tersebut selanjutnya digunakan pada proses *image text swapping*, yaitu menggantikan teks Mandarin pada gambar produk dengan teks hasil terjemahan sehingga informasi produk dapat dipahami secara langsung oleh pengguna ketika mengakses halaman 1688.com.

2.14 Penelitian Terdahulu

Sebelum memulai penelitian ini, penulis mencari referensi untuk melakukan penelitian dengan studi literatur mengenai penelitian terkait yang telah dilakukan oleh peneliti sebelumnya. Berikut adalah tabel yang berisikan beberapa penelitian terdahulu yang digunakan untuk penulis menjadi referensi dalam melakukan penelitian.

Tabel 2.1 Penelitian Terdahulu

No	Peneliti dan Tahun	Fokus Penelitian	Temuan Utama	Keterkaitan dengan Penelitian Ini
1	Chandra & Tiwari (2021)	*Web Scraping* berbasis DOM pada aplikasi web dinamis berbasis JavaScript.	Ekstraksi data menggunakan *DOM Scraping* terbukti lebih adaptif dan presisi pada halaman web dinamis berbasis JavaScript dibandingkan teknik *static HTML fetching*.	Menjadi dasar penggunaan teknik *DOM Scraping* via *Content Script* untuk mengekstraksi data produk pada platform 1688.com.
2	Chen et al. (2022)	*Pipeline* pengolahan citra (*OCR* dan	Penggabungan *deep learning OCR* dan teknik *inpainting/text*	Menjadi landasan penyusunan *pipeline* pemrosesan visual (*EasyOCR* dan

No	Peneliti dan Tahun	Fokus Penelitian	Temuan Utama	Keterkaitan dengan Penelitian Ini
		Image Text Swapping) untuk lokalisasi teks.	*overlay* mampu mengganti teks visual pada citra tanpa merusak tekstur latar belakang (*background*).	*Image Text Swapping*) untuk mentransformasi teks Mandarin pada gambar produk 1688.
3	Klosowski et al. (2022)	Optimasi penjadwalan *task/scheduler* pada arsitektur pemrosesan asinkron menggunakan *Firefly Algorithm*.	*Firefly Algorithm* efektif meminimalkan waktu penyelesaian total (*makespan*) dan mencegah *bottleneck* dalam penjadwalan antrean *task* paralel.	Menjadi dasar penerapan *Firefly Algorithm* untuk mengoptimalkan urutan eksekusi tugas (*Pipeline Scheduler*) di *backend server*.
4	Lam (2021)	Pengembangan *Chrome Extension* menggunakan arsitektur Manifest V3 untuk ekstraksi data *real-time*.	Arsitektur Manifest V3 melalui *Content Script* dan *Service Worker* memberikan akses *native* ke *DOM client* secara efisien tanpa konsumsi memori *server* yang besar.	Menjadi rujukan dalam membangun antarmuka dan mekanisme *client-side processing* berbasis *Chrome Extension Manifest V3*.

No	Peneliti dan Tahun	Fokus Penelitian	Temuan Utama	Keterkaitan dengan Penelitian Ini
5	Zhao & Wang (2023)	Sistem otomatisasi pemrosesan produk *e-commerce* mencakup *scraping*, *penerjemahan*, dan *OCR*.	Integrasi *scraping*, *penerjemahan*, dan *OCR* mampu mempercepat lokalisasi katalog produk, namun performa sistem menurun saat menangani antrean pemrosesan visual berukuran besar.	Menjadi acuan sistem yang paling mirip. Penelitian ini menyempurnakan keterbatasan tersebut dengan menambahkan *Image Text Swapping* dan optimasi *Pipeline Scheduler* berbasis *Firefly Algorithm*.

Berdasarkan kajian penelitian terdahulu pada Tabel 2.1, belum ditemukan penelitian yang mengintegrasikan *Chrome Extension* berbasis *DOM Scraping* dengan *pipeline* OCR, *Google Translate, Image Text Swapping*, serta optimasi *Pipeline Scheduler* menggunakan *Firefly Algorithm* secara terpadu pada platform 1688.com.

Sebagian besar penelitian terdahulu hanya berfokus pada satu atau dua aspek secara terpisah seperti *scraping* halaman dinamis saja (Chandra & Tiwari, 2021), *image text swapping* secara independen (Chen et al., 2022), atau optimasi penjadwalan tugas tanpa penerapan pada *pipeline* citra multibahasa (Klosowski et al., 2022). Adapun penelitian yang paling mendekati (Zhao & Wang, 2023) telah mengintegrasikan *scraping*, penerjemahan, dan OCR, namun belum menerapkan teknik rekonstruksi citra (*image text swapping*) serta belum memiliki mekanisme optimasi penjadwalan *task* otomatis saat terjadi beban pemrosesan tinggi.

Oleh karena itu, kebaruan (*novelty*) dari penelitian ini terletak pada pengintegrasian seluruh komponen tersebut ke dalam satu ekosistem yang utuh. Penelitian ini memanfaatkan *Chrome Extension (Manifest V3)* sebagai *client* berbasis *DOM Scraping*, yang terhubung dengan *backend Flask* untuk menjalankan *pipeline* OCR, penerjemahan, dan *Image Text Swapping*, dengan alokasi urutan eksekusinya dioptimalkan secara dinamis oleh *Firefly Algorithm* pada *Pipeline Scheduler*. Integrasi ini dirancang untuk menghasilkan sistem penyusunan katalog produk multibahasa dari platform 1688.com yang efisien dan adaptif.