Laporan Teknis Arsitektur Sistem Early Warning System: Integrasi Machine Learning dan Deployment API

MACHINE LEARNING

Disusun Oleh : Kelompok 4

Ghoffar Abdul Ja'far 2341720035/TI3H

Muhammad Afif Al Ghifari 2341720168/TI3H

Muhammad Irsyad Dimas Abdillah 2341720088/TI3H

Oktavian Eka Ramadhan 2341720117/TI3H

JURUSAN TEKNOLOGI INFORMASI

POLITEKNIK NEGERI MALANG

2025/2026

1. Pendahuluan

Dokumen ini menjelaskan rancangan dan implementasi **Early Warning System (EWS)** yang digunakan untuk memprediksi risiko keterlambatan pembayaran. Sistem dibangun dengan pendekatan **MLOps**, di mana proses pelatihan model dilakukan di lingkungan terpisah dari sistem produksi yang menangani prediksi secara waktu nyata.

Tujuan utama EWS ini adalah mengolah data transaksi mentah menjadi **skor risiko numerik (Risk_Score)** dan **kategori risiko** yang mudah dipahami serta dapat langsung digunakan sebagai dasar pengambilan keputusan. Untuk mencapai tujuan tersebut, sistem menggabungkan alur pengolahan data yang terstruktur, model **Stacking Ensemble**, serta layanan **API FastAPI** yang cepat dan efisien. Penyusunan laporan ini mengacu pada dokumentasi pengembangan model dan dokumentasi operasional API.

Dari sisi operasional, sistem dirancang agar siap digunakan di lingkungan produksi dengan kebutuhan ketersediaan tinggi. FastAPI dimanfaatkan untuk mendukung pemrosesan asinkron, sementara algoritma berbasis gradient boosting dipilih karena performa dan efisiensinya. Secara keseluruhan, pendekatan yang digunakan menekankan kejelasan proses, validasi data yang kuat, serta struktur kode yang modular agar sistem mudah dikembangkan dan dipelihara dalam jangka panjang.

2. Lingkungan dan Teknologi

Infrastruktur teknis sistem ini dikembangkan menggunakan ekosistem Python dengan memanfaatkan berbagai library yang umum digunakan di industri untuk komputasi numerik, pemodelan statistik, dan penyediaan layanan web yang cepat. Pemilihan teknologi tersebut disesuaikan dengan kebutuhan agar proses pelatihan model dapat direproduksi dengan baik, sekaligus memastikan waktu respons yang rendah pada saat sistem melakukan inferensi.

2.1 Spesifikasi Lingkungan Pengembangan (Development Environment)

Proses pengembangan model, rekayasa fitur, dan pengujian algoritma dilakukan dalam lingkungan komputasi interaktif, seperti Jupyter Notebook atau Google Colab. Lingkungan ini dikonfigurasi menggunakan manajer paket **pip** dengan sejumlah dependensi utama yang disesuaikan untuk kebutuhan eksperimen dan evaluasi model.

- **Runtime Utama:** Python 3.10 atau versi yang lebih baru.

- **Komputasi Numerik dan Manipulasi Data:**
  - pandas: Digunakan sebagai tulang punggung manipulasi data tabular, memungkinkan operasi *vectorized* pada dataset transaksi dan demografi.
  - numpy: Menyediakan dukungan untuk array multi-dimensi dan fungsi matematika tingkat rendah yang mendasari operasi *pandas* dan *scikit-learn*.
- **Framework Machine Learning:**
  - scikit-learn: Berperan sebagai kerangka kerja utama untuk *preprocessing*, validasi silang (*cross-validation*), kalkulasi metrik evaluasi (MAE, RMSE, R^2), dan implementasi meta-learner (*RandomForestRegressor*).
  - xgboost: Pustaka *eXtreme Gradient Boosting* yang digunakan untuk menangkap pola non-linear kompleks dengan efisiensi komputasi tinggi.
  - catboost: Algoritma *gradient boosting* yang dioptimalkan untuk menangani fitur kategorikal secara native dan mengurangi risiko *overfitting* melalui skema pemrosesan *ordered boosting*.
  - lightgbm: Implementasi *gradient boosting* berbasis pohon yang menggunakan teknik *Gradient-based One-Side Sampling* (GOSS) untuk kecepatan pelatihan dan efisiensi memori yang superior.

2.2 Spesifikasi Lingkungan Produksi (Production Environment)

Lingkungan ini dirancang untuk melayani permintaan inferensi melalui HTTP API. Spesifikasi lingkungan produksi meliputi:

- **Web Framework:** FastAPI. Dipilih karena dukungannya terhadap standar ASGI (*Asynchronous Server Gateway Interface*), validasi data otomatis menggunakan Pydantic, dan kemampuannya menghasilkan dokumentasi OpenAPI secara otomatis.
- **Server Aplikasi (ASGI):** Uvicorn. Berfungsi sebagai server ASGI *lightning-fast* yang menjalankan aplikasi FastAPI.
- **Dependency Management:** Seluruh pustaka produksi didefinisikan dalam berkas requirements.txt dan diinstal menggunakan versi pip terbaru untuk memastikan keamanan dan stabilitas dependensi.

2.3 Arsitektur Direktori Sistem

Organisasi berkas dalam repositori ML_INFERENCE_API menerapkan prinsip *Separation of Concerns* (SoC), memisahkan logika aplikasi, konfigurasi sistem, definisi skema data, dan artefak model. Struktur direktori ini memfasilitasi navigasi kode dan isolasi komponen:

- **ML_INFERENCE_API/**: Direktori akar proyek.

- **app/**: Modul inti aplikasi.
  - main.py: Titik masuk (*entry point*) aplikasi. Berkas ini menginisialisasi instansi FastAPI, mendefinisikan *middleware*, dan memetakan rute (*routes*) URL ke fungsi penanganan.
  - config.py: Modul manajemen konfigurasi. Berkas ini memuat variabel lingkungan dan mendefinisikan konstanta sistem seperti versi aplikasi (APP_VERSION) dan ambang batas risiko (*risk thresholds*).
  - **models_ews/**: Definisi struktur data.
    - schemas.py: Berisi kelas-kelas Pydantic yang mendefinisikan kontrak data untuk *request* (input) dan *response* (output), serta melakukan validasi tipe data secara ketat.
  - **services/**: Lapisan logika bisnis dan orkestrasi model.
    - model_loader.py: Bertanggung jawab untuk memuat (*loading*) artefak model .pkl dan metadata JSON dari disk ke dalam memori RAM saat *startup*.
    - feature_builder.py: Implementasi logika transformasi data. Modul ini mereplikasi proses rekayasa fitur yang dilakukan saat pelatihan, memastikan konsistensi antara data latih dan data inferensi.
    - predictor.py: Mengelola alur prediksi, termasuk pemanggilan model dasar (*base models*) dan agregasi hasil oleh meta-model (*stacking logic*).
  - **utils/**: Utilitas pendukung.
    - risk_analyzer.py: Melakukan pasca-pemrosesan (*post-processing*) terhadap skor numerik hasil prediksi, mengonversinya menjadi kategori risiko, dan menautkannya dengan rekomendasi bisnis.
- **models_ews/**: Penyimpanan persisten untuk artefak model produksi.
  - **Regression/**: Sub-direktori khusus untuk model regresi.
    - *Model Artifacts*: gb_regressor.pkl, rf_regressor.pkl, meta_ridge.pkl.
    - *Metadata*: model_info.json (informasi fitur), evaluation.json (metrik performa).
    - *Analysis*: gb_feature_importance.csv, rf_feature_importance.csv, model_comparison.csv.
- requirements.txt: Daftar dependensi paket Python yang diperlukan untuk lingkungan produksi.

- README.md: Dokumentasi operasional untuk *deployment* dan penggunaan API.

3. Pengembangan Model Machine Learning

Bagian ini menjelaskan pendekatan teknis yang digunakan untuk mengubah data mentah menjadi model prediktif yang andal, sebagaimana dijabarkan dalam README_PBL_REGRESSION. Proses tersebut meliputi tahap penyiapan dan pembersihan data, rekayasa fitur secara menyeluruh, serta penerapan strategi pelatihan model ansambel bertingkat untuk meningkatkan akurasi dan stabilitas prediksi.

3.1 Sumber Data dan Strategi Pembagian

Pengembangan model didasarkan pada dataset komposit yang menggabungkan data transaksi real-world dengan data demografis. Integritas proses pelatihan dijaga melalui strategi pemisahan data yang ketat.

- **Komposisi Dataset:**
  - transaction_data(RealData).csv: Sumber data primer berisi log transaksi historis.
  - warga_mapping.csv: Data sekunder yang menyediakan konteks demografis nasabah.
  - dataset_feature_engineered.csv: Dataset final hasil penggabungan dan rekayasa fitur yang menjadi input langsung bagi algoritma pembelajaran.
  - dataset_sintetis.csv: Dataset tambahan yang digunakan untuk augmentasi data atau pengujian stres model.
- **Mekanisme Pembagian Data (Splitting Strategy):**
  - Untuk mencegah kebocoran informasi (*data leakage*) dan memastikan evaluasi yang objektif, dataset dibagi menjadi **Data Latih (80%)** dan **Data Uji (20%)**.
  - Konsistensi pembagian dijamin melalui penggunaan *seed* pengacak statis (*Random State*) bernilai **42**. Hal ini memungkinkan reproduksi hasil eksperimen yang identik di masa mendatang.

3.2 Rekayasa Fitur (Feature Engineering)

Kinerja model sangat dipengaruhi oleh bagaimana data mentah diolah menjadi fitur yang relevan. Oleh karena itu, sistem ini menerapkan proses rekayasa fitur secara menyeluruh hingga

menghasilkan 27 fitur yang kemudian dikelompokkan ke dalam lima kategori analisis.

Tabel berikut merinci spesifikasi teknis dari setiap kelompok fitur:

Kategori Fitur	**Jumlah**	**Variabel Fitur**	**Mekanisme Ekstraksi & Signifikansi Teknis**
Temporal	8	Month, Day, DayofWeek, Quarter, Is_Weekend, Is_Month_End, Is_Month_Start, Days_From_Start	Fitur-fitur ini diekstraksi dari stempel waktu (*timestamp*) transaksi. Tujuannya adalah menangkap pola musiman (sirkadian, mingguan, bulanan) dalam perilaku pembayaran. Misalnya, Is_Month_End sering berkorelasi dengan siklus penggajian.
Behavioral	7	Total_Transactions, Avg_Nominal, Transaction_Frequency, Active_Duration_Days, Avg_Interval_Days, Delay_Count, Delay_Percentage	Menangkap profil risiko perilaku nasabah. Metrik seperti Delay_Percentage memberikan sinyal historis langsung terhadap risiko keterlambatan masa depan, sementara Avg_Interval_Days mengukur keteraturan transaksi.
Transaction Type	6	TopUp_Count, QRIS_Count,	Mengukur preferensi penggunaan layanan. Rasio

		Transfer_Count, TopUp_Ratio, QRIS_Ratio, Transfer_Ratio	penggunaan fitur tertentu (misal: Transfer vs QRIS) dapat menjadi indikator profil likuiditas atau gaya hidup nasabah.
Activity	2	Activity_Month, Activity_Quarter	Agregasi tingkat aktivitas pada jendela waktu tertentu, memberikan konteks tren jangka pendek dan menengah.
Nominal	1	Nominal_Transaksi	Nilai moneter dari transaksi itu sendiri, yang sering kali memiliki korelasi non-linear dengan risiko (misal: transaksi bernilai sangat besar mungkin memiliki profil risiko berbeda).

Eliminasi Fitur (Feature Selection):

Tahap praproses data juga mencakup penghapusan kolom yang tidak relevan atau berpotensi menimbulkan kebocoran data. Variabel seperti **No_Reff**, **Nama_Penerima**, dan **Nama_Pengirim** dihilangkan karena hanya berfungsi sebagai pengenal unik dan tidak memberikan nilai prediktif yang dapat digeneralisasi. Selain itu, variabel **Risk_Score** sebagai target dipisahkan dari seluruh fitur yang digunakan sebagai input model.

3.3 Arsitektur Model: Stacking Ensemble Regressor

Sistem ini tidak menggunakan satu algoritma saja, melainkan menerapkan pendekatan **Stacking Ensemble**. Metode ini mengkombinasikan hasil prediksi dari beberapa model dasar

(*base learners*) yang telah terbukti kuat, sehingga mampu menghasilkan prediksi akhir yang lebih akurat dan konsisten. Arsitektur yang digunakan disusun dalam dua lapisan utama.

3.3.1 Layer 0: Base Learners

Lapisan pertama pada arsitektur ansambel ini terdiri dari beberapa model prediktif yang dilatih secara terpisah pada dataset yang sama. Tujuan utama dari penggunaan lebih dari satu model pada lapisan ini adalah menciptakan keragaman prediksi, sehingga varians keseluruhan sistem dapat ditekan dan hasil inferensi menjadi lebih stabil.

1. **Gradient Boosting Regressor (GB):**
    - Model ini berfungsi sebagai salah satu prediktor utama yang menangkap hubungan non-linear antar fitur. Konfigurasinya dioptimalkan untuk keseimbangan antara bias dan varians, serta memberikan performa yang konsisten sebagai fondasi ansambel.
2. **Random Forest Regressor (RF):**
    - Model ini digunakan untuk melengkapi Gradient Boosting dengan pendekatan berbasis *bagging*, sehingga mampu meningkatkan robustitas prediksi terhadap noise dan variasi data.

3.3.2 Layer 1: Meta-Model

Pada lapisan kedua, sistem menggunakan Ridge Regression sebagai meta-model yang disimpan dalam berkas meta_ridge.pkl. Model ini bertugas mempelajari kombinasi optimal dari keluaran model dasar untuk menghasilkan prediksi akhir. Pendekatan ini dipilih karena stabilitas Ridge Regression dalam mengelola multikolinearitas antar prediksi base learner.

- **Algoritma: Ridge Regressor**.
- **Mekanisme:** Model ini bertugas mempelajari kombinasi optimal dari keluaran model dasar untuk menghasilkan prediksi akhir. Pendekatan ini dipilih karena stabilitas Ridge Regression dalam mengelola multikolinearitas antar prediksi base learner..

4. Penyimpanan dan Ekspor Model

Pasca pelatihan, seluruh komponen model dan metadata yang relevan diekspor ke sistem file untuk persistensi. Proses ini memastikan bahwa model yang di-deploy di API adalah replika

eksak dari model yang telah divalidasi. Seluruh artefak disimpan secara terstruktur dalam direktori ./models_ews/

4.1 Artefak Model (.pkl)

Model-model diekspor menggunakan format serialisasi biner Python (pickle). Format ini memungkinkan objek Python kompleks (termasuk arsitektur pohon keputusan dan bobot model) disimpan dan dimuat ulang secara efisien.

- **gb_regressor.pkl**: Merepresentasikan komponen model Gradient Boosting.
- **rf_regressor.pkl**: Merepresentasikan komponen model Random Forest.
- **meta_ridge.pkl**: Merepresentasikan model meta (Stacking Aggregator).

4.2 Metadata dan Dokumentasi Kinerja

Sistem penyimpanan juga mencakup file metadata yang krusial untuk operasional API dan auditabilitas model.

- **model_info.json**: Berkas ini berisi "cetak biru" model, mencakup daftar urutan fitur (*feature names*) yang wajib dipatuhi oleh input API. Ketidakcocokan urutan fitur dapat menyebabkan degradasi performa yang parah. Berkas ini juga menyimpan statistik pelatihan (seperti mean dan standar deviasi) dan stempel waktu pelatihan.
- **Analisis Kepentingan Fitur**: File gb_feature_importance.csv dan rf_feature_importance.csv mendokumentasikan kontribusi relatif setiap fitur terhadap keputusan model. Data ini penting untuk *explainability* (penjelasan) model dan debugging jika terjadi anomali prediksi.
- **model_comparison.csv**: Tabel komparasi performa antar model dasar, yang memvalidasi keputusan penggunaan arsitektur ensemble.

5. Implementasi Model pada FastAPI

Bagian ini menjelaskan arsitektur perangkat lunak yang digunakan untuk menyajikan model machine learning sebagai layanan web yang dapat diakses oleh sistem lain. Implementasi yang diterapkan menitikberatkan pada kinerja layanan, validasi data yang ketat, serta kemudahan integrasi dengan aplikasi atau layanan eksternal.

5.1 Mekanisme Pemuatan Model (Model Loading Strategy)

Untuk meminimalkan latensi respons pada setiap permintaan, sistem menerapkan strategi pemuatan model *eager loading*.

- **Layanan model_loader.py**: Modul ini bertanggung jawab untuk membaca artefak .pkl dan JSON dari disk.
- **Startup Event**: Proses pemuatan dipicu secara otomatis saat aplikasi FastAPI dimulai (event startup). Objek model dimuat ke dalam memori global aplikasi. Hal ini memastikan bahwa model siap melayani permintaan segera setelah server aktif, menghilangkan *overhead* I/O disk pada setiap transaksi inferensi.
- **Verifikasi Integritas**: Saat memuat, sistem juga memverifikasi keberadaan file metadata untuk memastikan bahwa model yang dimuat memiliki definisi fitur yang lengkap.

5.2 Skema Data dan Validasi (Pydantic)

FastAPI menggunakan pustaka Pydantic untuk mendefinisikan skema data dan melakukan validasi tipe secara otomatis. Ini adalah lapisan pertahanan pertama terhadap data input yang korup atau tidak valid.

5.2.1 Skema Input (Request Model)

API mengharapkan data transaksi mentah, bukan data yang sudah diolah.

- **tanggal** (*String*): Format ISO-8601 "YYYY-MM-DD" (Contoh: "2025-01-15"). Validasi format tanggal dilakukan untuk mencegah kesalahan parsing pada tahap rekayasa fitur.
- **nominal** (*Integer/Number*): Nilai transaksi moneter (Contoh: 500000). Harus berupa nilai numerik non-negatif.
- **target_type** (*String*): Kategori tujuan transaksi (Contoh: "broadcast", "payment").
- **rt_number** (*Optional*): Atribut pelengkap yang dapat bernilai null atau tipe data valid lainnya.

5.2.2 Skema Output (Response Model)

Struktur respons dirancang untuk memberikan informasi yang lengkap namun terstruktur kepada klien.

- **success** (*Boolean*): Indikator status eksekusi permintaan.
- **data** (*Object*):
  - **risk_score** (*Float*): Skor prediksi kontinu (0-100).
  - **risk_category** (*Object*): Hasil interpretasi bisnis.
    - status: Label tekstual (misal: "SEDANG", "TINGGI").
    - emoji: Indikator visual cepat (misal: "⚠️").
    - rekomendasi: Saran mitigasi deskriptif.
    - tindakan: Daftar (*list*) langkah operasional konkret yang disarankan.
  - **input_data**: Mengembalikan data input asli untuk keperluan verifikasi dan *logging* di sisi klien.

6. Alur Inferensi Sistem

Proses inferensi (Inference Pipeline) merupakan rangkaian tahapan yang mengubah data input mentah menjadi prediksi risiko akhir. Alur ini dirancang secara linear namun tetap modular, sehingga setiap komponen memiliki tanggung jawab yang jelas dan mudah dipelihara. Seluruh proses inferensi dijalankan sepenuhnya di sisi server (backend).

6.1 Tahapan Pemrosesan Data

1. **Penerimaan dan Validasi Permintaan**:
   - Klien mengirimkan HTTP POST ke endpoint /predict dengan *payload* JSON.
   - Pydantic memvalidasi tipe data. Jika input tidak sesuai (misal: nominal berupa string huruf), API menolak permintaan dengan kode status 422 Unprocessable Entity dan detail kesalahan yang spesifik.
2. **Konstruksi Fitur (On-the-fly Feature Engineering)**:
   - Data valid diteruskan ke layanan app/services/feature_builder.py.
   - **Transformasi Kritis**: Ini adalah tahap paling krusial dalam inferensi. Karena input adalah data mentah, API harus mereplikasi logika *preprocessing* yang kompleks dari tahap pelatihan.
   - Layanan ini mengurai string tanggal menjadi komponen temporal (Month, Day, Is_Weekend, dll).
   - Layanan menggabungkan input nominal dan target_type dengan fitur temporal hasil

ekstraksi.
    - Hasil akhirnya adalah vektor fitur berdimensi 27 yang struktur dan urutannya **persis** sama dengan data yang digunakan saat pelatihan model. Ketidaksesuaian di tahap ini akan fatal bagi akurasi prediksi.
3. **Prediksi Model Dasar (Layer 0)**:
    - Vektor fitur dikirim ke app/services/predictor.py.
    - Sistem memanggil metode predict() dari model gb_regressor dan rf_regressor (atau kombinasi base models yang aktif) secara sekuensial atau paralel.
    - Hasilnya adalah himpunan skor prediksi mentah dari masing-masing model dasar.
4. **Agregasi Meta-Model (Layer 1)**:
    - Skor prediksi dari Layer 0 disusun menjadi vektor input baru (metode *stacking*).
    - Meta-model (meta_ridge.pkl) memproses vektor ini. Karena meta-model telah dilatih untuk menyeimbangkan kelemahan masing-masing base model, ia menghasilkan satu nilai Risk_Score final yang terkalibrasi.
5. **Analisis Risiko dan Pasca-Pemrosesan**:
    - Skor mentah (0-100) diteruskan ke app/utils/risk_analyzer.py.
    - Modul ini mengevaluasi skor terhadap ambang batas (thresholds) yang dikonfigurasi di config.py.
    - Berdasarkan posisi skor, sistem menetapkan label kategori (Aman/Waspada/Bahaya) dan memilih template rekomendasi serta tindakan yang sesuai dari basis pengetahuan internal.
6. **Konstruksi Respons**:
    - Hasil analisis dikemas dalam struktur JSON standar Response Model dan dikirimkan kembali ke klien dengan kode status 200 OK.

6.2 Mode Debugging (Verbose)

Sistem juga menyediakan endpoint **/predict/verbose** yang ditujukan untuk kebutuhan diagnostik. Alur pemrosesannya identik dengan endpoint prediksi standar, namun respons yang dihasilkan dilengkapi dengan metadata internal, termasuk nilai prediksi mentah dari setiap model dasar sebelum dilakukan agregasi. Fitur ini memungkinkan tim data scientist untuk memantau dan menganalisis kinerja masing-masing komponen model secara lebih mendalam di lingkungan

produksi.

7. Pengujian dan Validasi API

Untuk menjamin keandalan sistem dalam lingkungan produksi, API dilengkapi dengan serangkaian endpoint utilitas yang mendukung pemantauan kesehatan (*health monitoring*) dan verifikasi konfigurasi.

7.1 Mekanisme Health Check

Endpoint GET /health menyediakan mekanisme detak jantung (*heartbeat*) bagi sistem orkestrasi (seperti Kubernetes atau Load Balancer).

- **Fungsi**: Memeriksa apakah proses server berjalan dan, yang lebih penting, memverifikasi variabel status global models_loaded.
- **Respon**: JSON { "status": "ok", "models_loaded": true }. Jika model gagal dimuat saat startup, status ini akan merefleksikan kegagalan tersebut, memungkinkan sistem manajemen infrastruktur untuk me-restart *container* atau layanan.

7.2 Introspeksi Model

Endpoint GET /models/info memungkinkan audit runtime terhadap model yang aktif.

- **Transparansi**: Mengembalikan daftar nama model yang dimuat (misal: ['gb_regressor', 'rf_regressor', 'meta_ridge']) dan daftar nama fitur (feature_names) yang diharapkan oleh model.
- **Validasi Konsistensi**: Endpoint ini memungkinkan pengembang untuk memverifikasi bahwa versi model yang berjalan di produksi adalah versi yang benar dan bahwa logika feature_builder menghasilkan fitur yang sesuai dengan ekspektasi model.

7.3 Pengujian Integrasi Inferensi

Validasi fungsional dilakukan dengan mensimulasikan permintaan dari klien. Payload pengujian dikirim ke endpoint **/predict** untuk memastikan seluruh alur pemrosesan—mulai dari validasi data menggunakan Pydantic, pembentukan fitur, hingga proses inferensi model—berjalan dengan benar tanpa menimbulkan kesalahan (*exception*). Pengujian dinyatakan berhasil apabila sistem mengembalikan respons JSON yang sesuai skema dan menghasilkan skor risiko dalam rentang

yang wajar, yaitu 0 hingga 100.