PENERAPAN SOFTWARE DESIGN PATTERN BERBASIS TEMPLATE DAN OBSERVER DALAM PELATIHAN MODEL XGBOOST NATIVE

Tjoargen Christoper Redja, Carudin
Informatika, Universitas Singaperbangsa Karawang
Jl. HS.Ronggo Waluyo, Puseurjaya, Telukjambe Timur, Karawang, Jawa Barat 41361
ogaimori653@gmail.com

ABSTRAK

Pelatihan model *machine learning*, khususnya dengan algoritma XGBoost, sering kali dilakukan menggunakan skrip yang monolitik dan imperatif, sehingga menyulitkan proses pengujian, pemeliharaan, serta penambahan fitur seperti early stopping, checkpointing, dan logging. Permasalahan ini menunjukkan belum diterapkannya prinsip rekayasa perangkat lunak modern dalam pengembangan pipeline pelatihan model. Penelitian ini mengusulkan pendekatan Trainer Loop Chain, yaitu kombinasi pola desain Template Method dan Observer untuk membangun arsitektur pelatihan model XGBoost yang modular, fleksibel, dan dapat diperluas. Pendekatan ini memisahkan logika pelatihan utama dari fitur tambahan melalui handler plug-in yang independen. Implementasi dilakukan menggunakan Python 3.13 dan library XGBoost, serta diuji pada dataset Breast Cancer Wisconsin Diagnostic dengan konfigurasi binary:logistic dan metrik logloss. Hasil pengujian menunjukkan bahwa nilai logloss pada data validasi berhasil diturunkan dari 0.46701 pada iterasi ke-0 menjadi 0.13808 pada iterasi ke-9, dan pelatihan dihentikan secara otomatis pada iterasi ke-13 oleh EarlyStopHandler dengan parameter patience = 2. Model disimpan secara berkala setiap lima iterasi menggunakan CheckpointHandler, dan proses pelatihan dipantau secara real-time oleh LoggerHandler tanpa perlu memodifikasi fungsi pelatihan utama. Pendekatan ini terbukti meningkatkan efisiensi eksperimen, keterbacaan kode, serta mendukung pengembangan pipeline machine learning yang berkelanjutan dan modular.

Kata kunci : *XGBoost, design pattern, template method, observer, modular training*

1. PENDAHULUAN

Perkembangan machine learning dalam beberapa tahun terakhir telah mendorong kemajuan signifikan dalam bidang prediksi, klasifikasi, dan analisis data. Salah satu algoritma yang menonjol dalam kompetisi maupun penerapan industri adalah XGBoost (Extreme Gradient Boosting), sebuah algoritma ensemble berbasis Gradient Boosting Decision Tree (GBDT) yang dirancang untuk efisiensi, skalabilitas, dan performa tinggi. XGBoost terbukti unggul di berbagai kompetisi seperti Higgs Machine Learning Challenge dan kontes Kaggle, serta banyak diterapkan dalam domain finansial, medis, dan sistem rekomendasi karena kemampuannya menghasilkan model prediktif yang akurat dan efisien [1]

Penelitian ini memilih fokus pada XGBoost karena merupakan algoritma boosting berbasis pohon yang paling banyak digunakan dalam praktik industri dan kompetisi, memiliki API yang mendukung ekstensi melalui callback, serta memiliki dokumentasi teknis yang memungkinkan penerapan desain modular. Dibanding algoritma lain seperti Random Forest atau SVM, XGBoost memberikan fleksibilitas tinggi untuk integrasi proses pelatihan yang disesuaikan, menjadikannya kandidat ideal untuk dieksplorasi dalam konteks software design pattern.

Secara teknis, XGBoost bekerja dengan membangun pohon keputusan secara iteratif, di mana setiap pohon baru bertujuan untuk memperbaiki kesalahan dari model sebelumnya. Model akhir merupakan hasil agregasi dari seluruh prediksi pohon, yang diperkuat secara bertahap melalui mekanisme boosting. Dalam implementasi modern seperti yang dikembangkan oleh Ou (2020), XGBoost telah mendukung pelatihan out-of-core pada GPU, yang memungkinkan pelatihan model terhadap dataset berskala besar melebihi kapasitas memori GPU. Optimalisasi ini menggunakan struktur data ELLPACK yang efisien dan teknik sampling berbasis gradien, sehingga memungkinkan efisiensi tinggi tanpa kehilangan akurasi [2]

Namun, dalam praktik pelatihan model, XGBoost sering diimplementasikan melalui class imperative monolitik yang mencampur seluruh langkah-langkah pelatihan, dimulai mulai dari memuat dataset, data pre-processing, training model, evaluation, hingga penyimpanan model dalam bentuk pickle, dalam satu blok kode besar. Pendekatan ini menimbulkan sejumlah permasalahan mendasar, antara lain, rendahnya modularitas untuk proses perkembangan kode selanjutnya, sulitnya reproducibility karena tidak adanya separation of concerns, dan fleksibilitas yang minim seperti untuk konfigurasi secara dinamis early stopping, learning rate, checkpointing dan logging [3]

Di sisi lain, rekayasa perangkat lunak modern memiliki prinsip dan praktik yang matang dalam mengatasi permasalahan seperti ini, salah satunya melalui penerapan Software Design Patterns. Design pattern seperti Template Method dan Observer telah terbukti efektif dalam meningkatkan struktur dan fleksibilitas sistem perangkat lunak. Template Method merupakan sebuah behavioral design pattern yang memungkinkan pendefinisian kerangka algoritma

secara tetap di dalam superclass, namun dengan beberapa bagian spesifik yang dapat dikustomisasi atau override oleh subclass, sementara Observer Pattern mendukung komunikasi antar objek dengan loose coupling, cocok untuk pemberitahuan dinamis dalam proses pelatihan model.

Sayangnya, integrasi design pattern ini ke dalam machine learning pipeline—terutama dalam konteks pelatihan XGBoost—masih belum banyak dikaji atau diterapkan secara formal, baik di lingkungan industri maupun akademik. Studi oleh Washizaki dkk. mengindikasikan bahwa meski pola desain perangkat lunak muncul di beberapa desain sistem Machine Learning, aplikasinya dalam pengembangan sistem yang modular, fleksibel, dan maintainable masih terbatas.

Berdasarkan permasalahan tersebut, penelitian ini mengusulkan pendekatan dalam bentuk arsitektur pelatihan modular untuk XGBoost yang menggabungkan pola Template Method sebagai kerangka dasar eksekusi dan pola Observer untuk mendukung penambahan fitur secara dinamis melalui handler modular. Dengan pendekatan ini, alur pelatihan dapat dikembangkan tanpa mengubah struktur utama, sekaligus mendukung integrasi fitur seperti logger, checkpoint, dan early stopping secara fleksibel.

Penelitian ini bertujuan untuk merancang arsitektur modular untuk pelatihan XGBoost berbasis design pattern Template Method dan Observer, mengimplementasikan bukti konsep dalam bentuk skrip Python yang mendukung handler dinamis, dan mengevaluasi kemudahan penggunaan, pengujian, dan pemeliharaan dibandingkan dengan pendekatan konvensional. Hasil penelitian ini diharapkan dapat menjembatani kesenjangan antara praktik machine learning engineering dan praktik software engineering yang terstruktur dan berkelanjutan dengan mengadopsi prinsip-prinsip rekayasa perangkat lunak dalam pengembangan sistem pelatihan model machine learning

2. TINJAUAN PUSTAKA

2.1. Penelitian Terdahulu

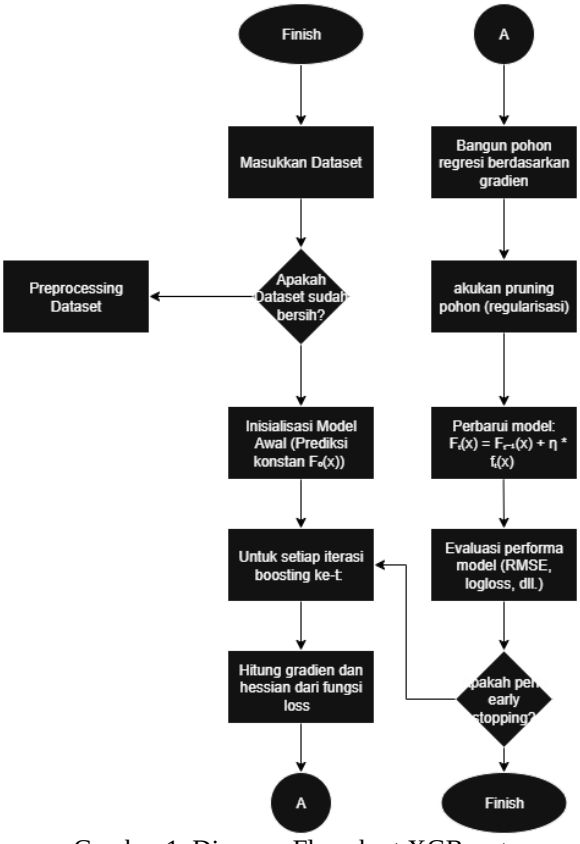
Penelitian terdahulu oleh Noor dkk. dalam artikel berjudul "XGBoost-Liver: An Intelligent Integrated Features Approach for Classifying Liver Diseases Using Ensemble XGBoost Training Model" mengembangkan model prediksi penyakit hati dengan memanfaatkan algoritma XGBoost yang diperkuat oleh kombinasi fitur dari berbagai teknik seleksi, seperti ANOVA, Chi-Square, PCA, dan LDA. Model ini juga menggunakan Fisher Score untuk seleksi fitur akhir, dan dievaluasi menggunakan k-fold cross-validation pada dataset ILPD (Indian Liver Patient Dataset). Hasilnya menunjukkan bahwa model XGBoost-Liver berhasil mencapai akurasi sebesar 92,07%, mengungguli berbagai metode pembanding lainnya seperti SVM, KNN, dan Random Forest [4]

Sculley et al. membahas bagaimana pipeline ML di perusahaan besar seperti Google dapat tumbuh menjadi sistem yang kompleks dan rapuh, jika tidak dikembangkan secara modular. Mereka menyebut istilah "technical debt in ML systems" dan mengadvokasi desain yang mengadopsi pattern seperti reusable trainer loops dan configurable evaluators [5]

Nielsen memberikan tinjauan mendalam tentang struktur eksternal XGBoost, menjelaskan aspek seperti objective functions, pruning, dan regularisasi, serta mendemonstrasikan bagaimana arsitektur ini bersifat fleksibel untuk diadaptasi oleh pipeline modular [6].

Persamaan dengan penelitian ini dengan topik milik Noor dkk terletak pada penggunaan algoritma XGBoost dan fokus pada optimalisasi performa pelatihan model machine learning. Namun, perbedaan utama dari penelitian ini adalah bahwa kami tidak fokus pada akurasi model dan langkah-langkah untuk meningkatkan performa model, melainkan pada perancangan arsitektur kerangka pelatihan yang modular dan fleksibel. Pendekatan yang diusulkan menerapkan pola desain Template Method dan Observer untuk memisahkan logika pelatihan utama dari fitur tambahan seperti logging, checkpointing, dan early stopping. Dengan demikian, penelitian kami memberikan kontribusi dalam menjembatani praktik machine learning dengan prinsip rekayasa perangkat lunak modern, yang belum dibahas dalam penelitian oleh Noor dkk.

2.2. XGBoost



Gambar 1. Diagram Flowchart XGBoost

XGBoost (eXtreme Gradient Boosting) adalah algoritma boosting berbasis pohon keputusan yang dirancang untuk efisiensi dan skalabilitas tinggi dalam

proses pelatihan model. Salah satu pengembangan terkini terhadap XGBoost diperkenalkan oleh Rong Ou (2020), yang merancang implementasi *out-of-core GPU gradient boosting* dalam pustaka XGBoost guna memungkinkan pelatihan pada dataset berskala besar yang melebihi kapasitas memori GPU tanpa menurunkan akurasi atau waktu pelatihan secara signifikan. Inovasi ini mengoptimalkan pemrosesan paralel GPU dengan strategi kompresi data (seperti ELLPACK) dan sampling berbasis gradien untuk efisiensi memori, memperkuat reputasi XGBoost sebagai pustaka machine learning yang cepat, hemat sumber daya, dan fleksibel dalam berbagai platform seperti Python, R, dan Java [2]

2.3. Software Design Pattern

Menurut Nesteruk dalam buku *Design Patterns in Modern C++20: Reusable Approaches for Object-Oriented Software Design*, pola desain perangkat lunak merupakan solusi modular yang dapat digunakan kembali untuk menyelesaikan masalah desain yang sering muncul dalam pengembangan perangkat lunak berorientasi objek. Pola ini tidak mengacu pada implementasi langsung, melainkan menyediakan struktur atau kerangka penyelesaian yang dapat diadaptasi terhadap berbagai konteks. Dalam pendekatan modern, setiap pola desain mencakup elemen seperti nama pola, permasalahan yang diselesaikan, solusi berbasis struktur kelas atau objek, serta konsekuensi dari penggunaannya dalam sistem yang lebih luas.

Template Method merupakan salah satu pola desain perilaku (behavioral design pattern) yang mendefinisikan alur atau kerangka dari suatu algoritma di dalam kelas induk, sementara rincian implementasi langkah-langkahnya didelegasikan ke kelas turunan. Pola ini menjaga struktur umum algoritma tetap konsisten, sambil memungkinkan variasi perilaku melalui override pada metode-metode tertentu. Menurut Nesteruk, pola ini sangat berguna dalam simulasi algoritmik seperti permainan papan, di mana kelas induk menentukan urutan langkah-langkah seperti start(), take_turn(), dan get_winner(), sedangkan implementasi detailnya disediakan oleh subclass. Dengan demikian, Template Method mendukung prinsip *inversion of control*, di mana kelas induk memanggil metode pada subclass, bukan sebaliknya, serta mendukung prinsip *Hollywood Principle*: "Don't call us, we'll call you."

Ciri-ciri utama dari Template Method:

a. Struktur algoritma ditentukan di kelas induk dan tidak dapat diubah oleh subclass.
b. Subclass dapat melakukan override terhadap metode-metode tertentu yang disebut *primitive operations*.
c. Mendukung prinsip pembalikan kontrol dan *open-closed principle*.
d. Dapat diimplementasikan menggunakan pendekatan imperatif berbasis class maupun pendekatan fungsional melalui *high-order functions* dan lambda expression.

Observer merupakan pola desain perilaku yang memisahkan antara objek pengirim notifikasi (subject) dan penerima notifikasi (observer), melalui mekanisme *subscribe–notify*. Menurut Nesteruk, pola ini memungkinkan sistem untuk merespon perubahan status secara dinamis tanpa ketergantungan eksplisit antar objek. Subjek menyimpan daftar observer, dan ketika terjadi perubahan data internal, subjek akan memanggil seluruh observer melalui metode notifikasi. Observer pattern mendukung arsitektur event-driven dan sangat umum diterapkan dalam sistem GUI, pemantauan sistem, atau arsitektur Model-View-Controller. Nesteruk juga menekankan pentingnya *thread-safety* serta penggunaan struktur tambahan seperti *decorator* untuk menjaga domain object tetap sederhana.

Ciri-ciri utama dari Observer Pattern:

a. Subjek menyimpan daftar observer yang dapat ditambahkan atau dihapus secara dinamis.
b. Subjek memanggil seluruh observer saat terjadi perubahan dengan metode notify().
c. Observer mengimplementasikan metode update() sebagai respon terhadap perubahan.
d. Mengurangi tight coupling antara pengirim dan penerima informasi.
e. Mendukung integrasi dengan event-loop, GUI, atau sistem real-time melalui arsitektur yang fleksibel dan loosely coupled.

2.4. Pola Desain dalam Sistem Machine Learning

Penelitian oleh Washizaki secara sistematis mengkaji literatur akademik maupun gray literature untuk mengidentifikasi praktik rekayasa perangkat lunak yang baik maupun buruk dalam bentuk pola desain (design pattern) dan antipola (anti-pattern) pada pengembangan sistem machine learning (ML). Dari hasil tinjauan, mereka merumuskan 12 pola arsitektur ML, 13 pola desain ML, serta 8 antipola yang diklasifikasikan berdasarkan kompleksitas dan skenario penggunaannya. Penelitian ini menunjukkan bahwa meskipun pola desain untuk sistem ML belum sepenuhnya terdokumentasi dalam format standar, namun memiliki potensi besar untuk meningkatkan modularitas, keterandalan, dan kemudahan pemeliharaan sistem ML dalam skala industri [7]

Lebih lanjut, kajian sistematis oleh Görkem Giray menguraikan tantangan dan pendekatan software engineering dalam merekayasa sistem ML secara holistik. Penelitian ini mengkaji 141 publikasi dari jurnal dan konferensi software engineering ternama, dan menemukan bahwa hampir semua aspek rekayasa perangkat lunak—mulai dari desain, pengujian, manajemen konfigurasi, hingga proses rekayasa perangkat lunak—terdampak secara signifikan oleh sifat non-deterministik dan data-driven dari sistem ML [8]

Giray mencatat bahwa teknik desain dan pengembangan tradisional perlu disesuaikan agar dapat mengakomodasi kebutuhan sistem ML yang dinamis, dan menyoroti kurangnya penerapan pola desain struktural seperti Template Method atau

Observer dalam siklus pelatihan dan pengelolaan model ML. Hal ini menunjukkan bahwa pengembangan loop pelatihan modular, seperti yang ditawarkan oleh penelitian ini, masih merupakan bidang yang sangat terbuka untuk eksplorasi akademik dan praktis [8].

Amershi et al. menyoroti praktik terbaik bagi tim machine learning, termasuk modularisasi, reusable components, dan pipeline yang dapat diuji terpisah — semua aspek tersebut mencerminkan ide arsitektur modular seperti implementasi hybrid antara metode *template* dan *observer* [9]

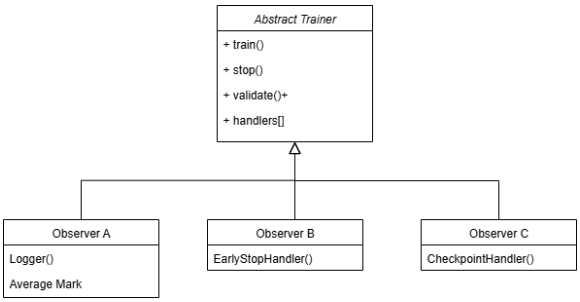
Király et al. menganalisis prinsip-prinsip dan pola arsitektural yang digunakan dalam desain toolbox machine learning seperti scikit-learn dan Weka. Mereka memperkenalkan kerangka 'scientific types' sebagai tipe formal untuk menangani data dan operasi statistik, serta merekomendasikan pattern seperti Template, Strategy, dan Composite sebagai dasar untuk membangun pipeline modular dan konsisten [10].

3. METODE PENELITIAN

Penelitian ini bersifat rekayasa eksperimental dengan fokus pada penerapan prinsip-prinsip software design pattern ke dalam proses pelatihan model XGBoost. Tujuannya adalah membangun kerangka kerja yang modular, dapat diperluas, dan mudah diuji ulang (testable).

Pendekatan ini merancang sebuah pola yang merupakan kombinasi dari *Template Method* untuk mendefinisikan struktur pelatihan, dan *Observer Pattern* untuk modularisasi *event* seperti *logging*, validasi, dan *checkpoint*.

3.1. Arsitektur Trainer Loop Chain



Gambar 2. Diagram Arsitektur pola Trainer Loop Chain

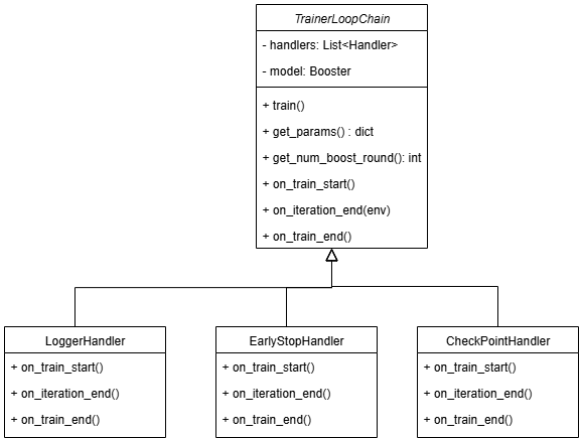
Pada gambar 1, penerapan gabungan Template Method dan Observer dibuat dalam bentuk diagram kelas menjadi 4 kelas, dengan kelas AbstractTrainer sebagai superclass yang menerapkan pola Template Method dan 3 kelas lainnya sebagai Observer, fungsi-fungsinya yaitu AbstractTrainer bertanggung jawab untuk kerangka pelatihan (train()). step() dan validate() membutuhkan subclass untuk menyediakan implementasi mereka masing-masing sebagia bentuk penerapan Template Method handlers[] berperan untuk menampung daftar observer. Handler seperti Logger, EarlyStopHandler, dan CheckpointHandler dipanggil pada event seperti awal epoch, akhir batch, dan seterusnya.

3.2. Tahapan Penelitian

Tahapan yang dilalui dalam penelitian ini mencakup:

a. Studi literatur dan analisis pola desain dalam konteks ML
b. Perancangan arsitektur Trainer Loop Chain
c. Perancangan Pseudocode TrainerLoopChain
d. Pemahaman Data dan Sumber Data
e. Implementasi dalam Python menggunakan xgboost, sklearn, dan logging
f. Evaluasi dengan membandingkan modularitas dan fleksibilitas pendekatan ini terhadap pendekatan konvensional (skrip monolitik XGBoost

3.3. *Desain Kelas (Pseudocode)*



Gambar 3. Diagram Arsitektur Pseudocode Trainer Loop Chain

Pola gabungan Template Method dan Observer ini diberikan nama TrainerLoopChain Method, dan berperan sebagai kelas utama di dalam pseudocode. Kelas ini berfungsi untuk memegang kerangka pelatihan (train()) yang tidak berubah, mengatur pemanggilan handler pada titik event (on_train_start, on_iteration_end, on_train_end), dan merupakan implementasi dari pola Template Method. Beberapa bagian dalam kelas utama bisa diubah tapi tetap mempertahankan struktur inti pelatihan.

LoggerHandler, EarlyStopHandler, CheckpointHandler merupakan implementasi pola Observer, di mana masing-masing handler mendengarkan dan merespon event yang dipancarkan oleh setiap loop pelatihan. Mereka tidak saling bergantung dan bisa didaftarkan atau dilepas secara dinamis. LoggerHandler berfungsi untuk mencatat log, EarlyStopHandler menghentikan pelatihan saat tidak ada peningkatan, dan CheckpointHandler menyimpan model terbaik.

3.4. Data dan Sumber Data

Dataset yang digunakan untuk penelitian ini berasal dari Breast Cancer Wisconsin yang dimuat ke dalam program lewat library scikit-learn. Fitur-fitur yang ada di dalam dataset dihitung dari gambar digital hasil pengambilan sampel jaringan (aspirasi jarum halus/FNA) pada benjolan di payudara. Dataset ini dikumpulkan pada tahun 1995, dengan 569 baris data dengan 30 fitur. Data ini akan digunakan dalam penerapan model XGBoost dengan pola desain perangkat lunak gabungan antara Template Method dan Observer. Dataset yang diambil relatif bersih tanpa ada missing values sama sekali, dan karena formatnya banyak berbentuk kontinyu atau float, maka tidak memerlukan encoding. Penelitian ini juga memanfaatkan sifat XGBoost yang berbasis pohon keputusan, sehingga tidak memerlukan normalisasi atau standarisasi fitur.

Tabel 1. Dataset untuk Training

Jenis Data	Jumlah Data
Sampel	569
Fitur	30

Tabel 2. Jenis Data Atribut

Jenis Data	Nama Field
Kategoris	ID
Kategoris	Diagnosis
Kontinyu	radius1
Kontinyu	texture1
Kontinyu	perimeter1
Kontinyu	area1
Kontinyu	smoothness1
Kontinyu	compactness1
Kontinyu	concavity1
Kontinyu	concave_points1
Kontinyu	symmetry1
Kontinyu	fractal_dimension1
Kontinyu	radius2
Kontinyu	texture2
Kontinyu	perimeter2
Kontinyu	area2
Kontinyu	smoothness2
Kontinyu	compactness2
Kontinyu	concavity2
Kontinyu	concave_points2
Kontinyu	symmetry2
Kontinyu	fractal_dimension2
Kontinyu	radius3
Kontinyu	texture3
Kontinyu	perimeter3
Kontinyu	area3
Kontinyu	smoothness3
Kontinyu	compactness3
Kontinyu	concavity3
Kontinyu	concave_points3
Kontinyu	symmetry3
Kontinyu	fractal_dimension3
Jenis Data	Jumlah Data
Sampel	569
Fitur	30

4. HASIL DAN PEMBAHASAN

4.1. Implementasi Sistem

Implementasi dilakukan dengan Python 3.13 menggunakan library xgboost, scikit-learn, pandas untuk pengecekan awal, dan sistem handler khusus untuk modularisasi proses pelatihan. Kode telah diorganisasi ke dalam tiga komponen utama, TrainerLoopChain yaitu nama kelas utama atau superclass yang mengatur siklus pelatihan, subclass yang merupakan implementasi Observer yaitu LoggerHandler, EarlyStopHandler, CheckpointHandler. Dataset ini dibagi menjadi 80% data pelatihan dan 20% data validasi, dengan dua kelas target yang harus diklasifikasi yaitu malignant dan benign.

Model XGBoost dilatih menggunakan fungsi xgb.train() dengan konfigurasi parameter objective sebagai binary:logistic dan eval_metric sebagai logloss. Pelatihan dilakukan selama 50 iterasi (num_boost_round=50). Sebagai bentuk penerapan pola Observer, fungsi callback logging_handler dipasang sebagai handler yang secara modular merekam nilai log-loss pada data validasi setiap iterasi, kemudian disimpan dalam list log_history. Dengan demikian, proses monitoring evaluasi model dilakukan secara modular tanpa mengubah struktur utama pelatihan.

```
class LoggerHandler(TrainingCallback):
    def before_training(self, model):
        print("[Logger] Pelatihan dimulai...")
        return model

    def after_training(self, model):
        print("[Logger] Pelatihan selesai.")
        return model

    def after_iteration(self, model, epoch, evals_log):
        val_logloss = evals_log['eval']['logloss'][epoch]
        print(f"[Logger] Iterasi {epoch}, logloss: {val_logloss:.5f}")
        return False  # continue training
```

Gambar 4. Cuplikan *LoggerHandler Class*

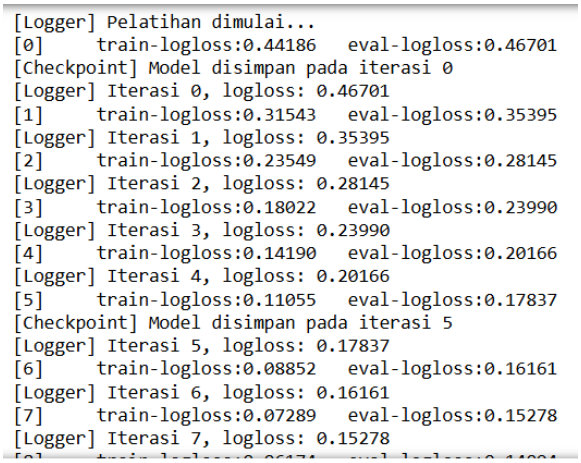
Potongan kode LoggerHandler pada gambar 4 berfungsi untuk mencetak log pada setiap iterasi tanpa memodifikasi fungsi train() utama dan terhubung dengan proses pelatihan XGBoost.

```
157 class CustomXGBoostTrainer(TrainerLoopChain):
158     def get_params(self):
159         return {
160             'objective': 'binary:logistic',
161             'eval_metric': 'logloss',
162             'verbosity': 0
163         }
164
165
```

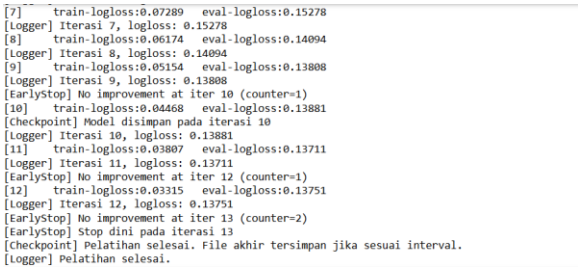
Gambar 5. Class CustomXGBoostTrainer

Sedangkan gambar 5 merupakan cuplikan program CustomXGBoostTrainer yang merupakan merupakan turunan dari TrainerLoopChain, yang bertugas mengembalikan konfigurasi parameter pelatihan XGBoost secara modular.

4.2. Hasil Pengujian



Gambar 6. *Proses Pelatihan Model XGBoost*



Gambar 7. Lanjutan *Proses Pelatihan Model XGBoost*

Pada gambar 6, saat pelatihan dimulai, LoggerHandler mencetak "[Logger] Pelatihan dimulai...", menandakan sistem telah memasuki fase inisialisasi. EarlyStopHandler mulai memantau metrik logloss dengan parameter patience = 2, dan mencetak informasi ketika tidak ada perbaikan pada iterasi tertentu. CheckpointHandler menyimpan model setiap 5 iterasi dan mencetak pesan status saat model disimpan. LoggerHandler menampilkan nilai logloss validasi secara real-time, misalnya: pada iterasi ke-0 sebesar 0.46701, kemudian berlanjut menurun hingga 0.13808 pada iterasi ke-9. Setelah iterasi ke-10 dan ke-12 pada gambar 7 tidak menunjukkan peningkatan berarti, EarlyStopHandler menghentikan pelatihan dini pada iterasi ke-13. Model akhir tersimpan otomatis sesuai interval oleh CheckpointHandler, dan LoggerHandler mencetak "[Logger] Pelatihan selesai." sebagai penutup siklus pelatihan.

Dalam penerapan gabungan pola desain template dan observer pada program pelatihan model XGBoost, tidak ada satu pun logika pelatihan yang harus dimodifikasi untuk menambahkan fungsionalitas seperti pencatatan dan penyimpanan model. Semua dilakukan melalui handler yang independen, yang mengikuti pola Observer dan hanya memonitor fase-fase tertentu dalam loop pelatihan. Jika suatu saat pengguna ingin menambahkan handler baru, seperti visualisasi grafis atau integrasi dengan TensorBoard, maka cukup menulis class baru dan menambahkannya ke parameter handlers tanpa menyentuh fungsi train() utama sama sekali.

Penerapan gabungan dua pola desain ini, yaitu Template Method dan Observer, meningkatkan efisiensi proses pelatihan model dari berbagai aspek. Pertama, efisiensi struktur kode diperoleh melalui pemisahan logika utama dan fitur tambahan, di mana handler seperti logging dan checkpointing dapat ditambahkan tanpa mengubah fungsi pelatihan inti. Kedua, efisiensi eksperimen ditingkatkan karena penambahan atau penggantian komponen cukup dilakukan melalui handler modular, sehingga mendukung eksperimen yang cepat dan fleksibel. Ketiga, efisiensi pemeliharaan juga meningkat karena setiap handler bersifat reusable dan loosely coupled, sehingga dapat digunakan ulang di berbagai proyek atau konfigurasi tanpa duplikasi kode. Dengan demikian, efisiensi yang dicapai tidak hanya bersifat komputasional, tetapi juga struktural dan fungsional dalam konteks pengembangan sistem machine learning yang berkelanjutan.

4.3. Analisis Hasil

Tabel 3. Perbandingan proses training model XGBoost Konvensional dengan Modular

Aspek yang Dianalisis	**Pelatihan Konvensional**	**Pola TrainerLoopChain**
Struktur kode	Monolitik	Modular, terpisah per fungsi
Logging/EarlyStop	Disisipkan manual	Tinggal *plug-in handler*
Pengujian tiap komponen	Sulit	Handler bisa diuji terpisah
Penambahan fitur baru	Harus ubah *loop*	Penambahan pada *handler*
Cocok untuk eksperimen cepat	Tidak	Iya

Dari Tabel 3 dapat dilihat bahwa pola TrainerLoopChain menawarkan berbagai keunggulan dibanding pendekatan pelatihan XGBoost konvensional. Pada aspek struktur kode, pendekatan konvensional cenderung bersifat monolitik, di mana seluruh logika pelatihan, evaluasi, dan logging tercampur dalam satu blok program. Hal ini menyebabkan keterbatasan dalam hal pemeliharaan kode dan kesulitan saat pengujian ulang (retesting). Sebaliknya, TrainerLoopChain memisahkan setiap fungsionalitas ke dalam handler modular, sehingga struktur kode menjadi lebih bersih dan terorganisasi.

Dalam hal penambahan fitur baru seperti logging, checkpoint, atau early stopping, pendekatan konvensional mengharuskan perubahan langsung pada bagian loop utama pelatihan. Ini meningkatkan risiko terjadinya bug dan mengganggu alur utama pelatihan. Dengan pendekatan modular berbasis pola Observer, penambahan fitur cukup dilakukan dengan menyisipkan handler baru tanpa menyentuh loop utama.

Keuntungan lain yang signifikan adalah kemudahan dalam eksperimen cepat. Dalam penelitian

dan pengembangan model, eksperimen iteratif sangat umum. TrainerLoopChain memungkinkan penggantian komponen seperti jenis logger atau metode evaluasi dengan cepat, mendukung siklus eksperimen yang lebih efisien.

Secara keseluruhan, hasil analisis menunjukkan bahwa pendekatan Trainer Loop Chain memiliki nilai praktis yang tinggi, terutama dalam proyek machine learning yang memerlukan pengembangan berkelanjutan, kolaboratif, dan berulang.

5. KESIMPULAN DAN SARAN

Penelitian ini berhasil merancang dan mengimplementasikan arsitektur modular untuk pelatihan model XGBoost berbasis pola desain Template Method dan Observer, yang disebut Trainer Loop Chain. Arsitektur ini dirancang untuk mengatasi permasalahan dalam pengembangan sistem machine learning, khususnya terkait modularitas, reusabilitas, dan fleksibilitas. Hasil implementasi dan pengujian terhadap dataset Breast Cancer Wisconsin menunjukkan bahwa Trainer Loop Chain mampu memisahkan logika pelatihan utama dari fitur tambahan seperti logging, checkpointing, dan early stopping secara independen tanpa memodifikasi struktur inti pelatihan. Pendekatan ini mendukung prinsip separation of concerns dan modularitas tinggi, memungkinkan handler seperti LoggerHandler dan EarlyStopHandler dikembangkan atau diuji secara terpisah. Selain itu, sistem ini meningkatkan efisiensi eksperimen karena penambahan fitur cukup dilakukan melalui plug-in handler, bukan dengan mengubah kode utama, sehingga memperkuat aspek maintainability dan extensibility dalam konteks rekayasa perangkat lunak modern. Dibandingkan pendekatan konvensional yang bersifat monolitik, Trainer Loop Chain memberikan keunggulan dalam hal struktur kode, keterbacaan, fleksibilitas eksperimen, serta potensi integrasi jangka panjang. Namun, penelitian ini masih terbatas pada penggunaan satu jenis algoritma (XGBoost) dan satu jenis dataset klasifikasi biner, sehingga belum dapat disimpulkan efektivitas pendekatan ini secara umum pada konteks lain. Oleh karena itu, untuk pengembangan selanjutnya, disarankan agar Trainer Loop Chain diuji pada berbagai algoritma lain seperti LightGBM, CatBoost, atau Neural Network, serta dataset yang lebih kompleks seperti multi-class dan regresi, guna mengevaluasi generalisasi desain arsitektur ini. Lebih jauh, arsitektur ini dapat dibungkus dalam pustaka open-source atau template proyek guna mendukung adopsi oleh komunitas machine learning, serta diterapkan dalam konteks ensemble dan AutoML untuk menguji skalabilitas pada sistem pelatihan berskala besar. Dengan pendekatan yang terstruktur dan berlandaskan prinsip-prinsip software engineering, Trainer Loop Chain diharapkan menjadi kontribusi awal yang penting dalam membangun pipeline machine learning yang kuat tidak hanya dari sisi akurasi, tetapi juga dari sisi arsitektur perangkat lunaknya.

DAFTAR PUSTAKA

[1] J. Verbraeken, M. Wolting, J. Katzy, J. Klop-Penburg, and J. S. Rellermeyer, “3 A Survey on Distributed Machine Learning.”

[2] R. Ou, “Out-of-Core GPU Gradient Boosting,” May 2020, [Online]. Available: http://arxiv.org/abs/2005.09148

[3] J. Liu *et al.*, “From Distributed Machine Learning to Federated Learning: A Survey,” Apr. 2021, doi: 10.1007/s10115-022-01664-x.

[4] S. Noor, S. A. Al Qahtani, and S. Khan, “XGBoost-Liver: An Intelligent Integrated Features Approach for Classifying Liver Diseases Using Ensemble XGBoost Training Model,” *Computers, Materials and Continua*, vol. 83, no. 1, pp. 1435–1450, 2025, doi: 10.32604/cmc.2025.061700.

[5] D. Sculley *et al.*, “Hidden Technical Debt in Machine Learning Systems.”

[6] D. Nielsen, “Tree Boosting With XGBoost Why Does XGBoost Win ‘Every’ Machine Learning Competition?”

[7] H. Washizaki *et al.*, “Software-Engineering Design Patterns for Machine Learning Applications,” *Computer (Long Beach Calif)*, vol. 55, no. 3, pp. 30–39, Mar. 2022, doi: 10.1109/MC.2021.3137227.

[8] G. Giray, “The pre-print of the A Software Engineering Perspective on Engineering Machine Learning Systems: State of the Art and Challenges.” [Online]. Available: https://conferences.oreilly.com/software-architecture/sa-eu-

[9] S. Amershi *et al.*, “Software Engineering for Machine Learning: A Case Study.” [Online]. Available: https://docs.microsoft.com/en-us/azure/devops/learn/devops-at-microsoft/

[10] F. J. Király, M. Löning, A. Blaom, A. Gueciоueur, and R. Sonabend, “Designing Machine Learning Toolboxes: Concepts, Principles and Patterns,” Jan. 2021, [Online]. Available: http://arxiv.org/abs/2101.04938