



Driving Test Result Prediction System

Rahul Hans¹, Deepanshi², Ms.Anju saini³

Department of CSE-AI&DS,

Panipat Institute of Engineering & Technology, Samalkha, India^{1,2,3}

³Mail ID: anju.cse.ai-ds@piet.co.in

Abstract— *Driving schools need practical ways to assess whether learners are ready for a formal driving test. Manual assessment can be subjective and may not consistently combine demographic, behavioral, and performance-related evidence. This paper presents a Driving Test Result Prediction System that uses machine learning to estimate a learner's likelihood of passing a driving test. The proposed system considers age, prior driving experience, first vehicle used for practice, practice hours, formal driving lessons, written test score, and self-reported confidence. The implementation integrates MySQL for data storage, preprocessing for feature preparation, a machine-learning model for prediction, FastAPI for backend services, Streamlit for the interactive interface, and Power BI for visualization. The workflow is intended to support early identification of learners who may need additional practice or targeted instruction. Because the supplied project synopsis does not report a completed dataset, a finalized learning algorithm, or measured test accuracy, this paper describes the proposed system and methodology without inventing experimental performance values.*

Keywords: Driving test prediction, machine learning, driving school, FastAPI, MySQL, Streamlit, Power BI, predictive analytics

INTRODUCTION

A driving test is intended to verify that a learner has developed sufficient knowledge, vehicle control, and confidence to operate a vehicle safely. Driving schools commonly monitor progress through instructor observations, practice sessions, written assessments, and informal feedback. Although these activities provide valuable information, their interpretation may vary between instructors and may be difficult to combine into a consistent readiness assessment. A data-driven prediction system can complement instructor judgment by organizing available learner information and producing a repeatable estimate of test readiness.

The Driving Test Result Prediction System is intended for driving schools and instructors. Its purpose is not to replace the official driving examination or professional instructor judgment, but to provide an additional predictive signal that can identify learners who may benefit from more practice, additional lessons, or targeted instruction. The proposed system uses demographic, behavioral, and performance-related attributes including age, prior experience, first vehicle used for practice, practice hours, formal lessons, written test score, and confidence level.

The system emphasizes an integrated software architecture. Student records are stored in MySQL, data are prepared for machine-learning processing, FastAPI exposes services for entering records and obtaining predictions, Streamlit supplies the user-facing web application, and Power BI supports



dashboard-based visualization. This separation of responsibilities makes the proposed solution suitable for an academic prototype and leaves room for future integration with external data sources.

RELATED WORK

Machine-learning systems are increasingly used to support prediction and decision-making from structured data. In the proposed driving-test application, the learning task differs from image or text classification because the input is primarily tabular information describing learner preparation. The relevant attributes are directly connected to the training process: previous experience, practice exposure, lesson attendance, written-test performance, and confidence. This makes careful feature definition and consistent data collection important.

A useful predictive workflow begins with historical observations for which the final test outcome is known. The historical records provide examples from which a supervised model can learn relationships between learner attributes and the target outcome. After training, a new learner record can be transformed using the same preprocessing operations and passed to the model. The result is a probability or score that can be presented to an instructor as an estimate of readiness.

The project also differs from a prediction-only prototype because it includes an application and analytics layer. MySQL provides persistent records, FastAPI exposes reusable services, Streamlit provides direct interaction, and Power BI provides a reporting view. This separation supports maintainability and makes the prediction component reusable. However, the value of the complete system still depends on data quality, consistent definitions, and evaluation on representative learner populations.

PROBLEM STATEMENT AND OBJECTIVES

The central problem is the difficulty of making consistent, data-supported judgments about whether a learner is likely to pass a driving test. A manual process can overlook relationships among several factors. For example, a learner with substantial practice hours but a low written-test score may require a different intervention from a learner with strong written performance but limited practical experience. A predictive system can combine these attributes into one estimated outcome and make the underlying information easier for instructors to review.

The primary objective is to create a comprehensive predictive system that determines whether a student is likely to pass a driving test from a combination of relevant learner attributes. A second objective is to reduce subjective bias in routine progress assessment by applying a consistent computational procedure. A third objective is to support personalized guidance, such as recommending additional practice sessions or focusing on specific driving skills.

At the organizational level, the project is intended to help driving schools improve pass rates, plan instructor availability, and allocate training resources more efficiently. The intended scope includes data capture, preprocessing, model training, prediction, API access, interactive frontend access, and dashboard visualization. By identifying at-risk learners early, the system can support corrective action before the formal test.

The system can be viewed as three cooperating layers: data management, predictive services, and user-facing analytics. The data layer is responsible for persistence and retrieval of learner information. The predictive layer performs validation, preprocessing, model inference, and result generation. The application and analytics layer presents the result to instructors and summarizes historical information.



This layered view helps define clear interfaces between components and reduces dependence on implementation details.

The database should preserve enough information to support later analysis without requiring the model to access unnecessary personal information. Training records and official test outcomes should be linked through stable internal identifiers. Prediction requests can also be recorded with the model version used at the time of inference. This allows a later review to distinguish between an old prediction and a result produced after the model was retrained.

The API layer provides a useful boundary for testing because the prediction service can be exercised independently of the Streamlit interface. Automated requests can verify that valid learner records generate valid responses and that invalid requests are rejected. The same API can later support another client, such as a mobile application, without changing the core prediction logic.

Visualization serves a different purpose from prediction. The prediction model answers an individual question about readiness, whereas the dashboard can reveal patterns across many learners. Keeping these roles separate prevents the dashboard from becoming a substitute for the prediction service and allows administrators to use aggregated information for planning while instructors use individual predictions for training decisions.

Scalability requirements are expected to be modest for an academic driving-school prototype, but the architecture should still avoid unnecessary coupling. If the number of learners grows, the database can remain the persistent source while the API and frontend continue to operate as separate services. Model retraining can be scheduled independently of day-to-day prediction requests, reducing the risk that a training operation interferes with normal application use.

SYSTEM OVERVIEW

The proposed solution follows a pipeline architecture. At the input stage, learner information is collected and stored in a relational database. The data-processing stage handles missing values, encodes categorical attributes, and normalizes numerical values. The processed records are supplied to a machine-learning model trained to estimate the likelihood of passing. The prediction is exposed through FastAPI endpoints and can be accessed through the Streamlit web interface. Power BI is used to visualize student-performance trends and areas requiring improvement.

This architecture separates data management, prediction services, user interaction, and visualization. MySQL acts as the persistent storage layer, FastAPI as the service layer, Streamlit as the application layer, and Power BI as the analytical layer. The separation also makes individual components easier to maintain and replace as the project develops.

METHODOLOGY

The methodology consists of sequential stages: data collection, data processing, model training, API integration, frontend development, and visualization. The workflow begins with data input, continues through data cleaning and model training, moves to prediction and visualization, and ends after recommendations are shown.

Each historical observation should contain the learner attributes available before the official driving test together with the eventual outcome used as the supervised-learning label. The project synopsis lists name, date of birth, prior driving experience, first vehicle, practice hours, driving lessons, written test score, and confidence level. Personal identifiers are useful for administration but should not be treated automatically

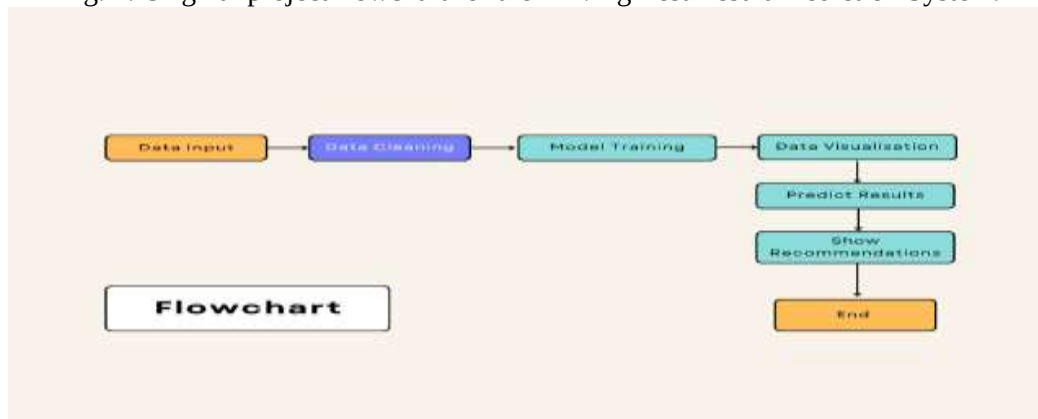


as predictive features. The modeling dataset should therefore distinguish operational identity fields from attributes intended for prediction.

Preprocessing converts raw records into a consistent model input. Missing numerical values can be handled using a documented imputation rule, while categorical fields such as prior experience or vehicle type require an explicit encoding scheme. Numerical variables such as practice hours, number of lessons, and written-test score should be represented consistently across training and inference. Feature engineering should remain traceable so that an instructor can understand which original learner information contributed to the prediction.

The synopsis states that a machine-learning model predicts the likelihood of passing and can improve as new test results are added. Because no final algorithm is specified in the source, the implementation should compare suitable supervised-learning algorithms using a fixed validation procedure. The selected model should be chosen from evidence such as validation performance, stability, interpretability, computational cost, and ease of integration with the backend.

Fig. 1. Original project flowchart for the Driving Test Result Prediction System.



SYSTEM ARCHITECTURE

FastAPI provides the backend API integration described in the project specification. Planned endpoints support adding new student data, requesting predictions, and retrieving test results. This service-oriented layer separates application logic from the user interface and provides a consistent way for the frontend or approved clients to communicate with the prediction system.

Streamlit provides the interactive web application through which users can access the system. A typical interaction consists of entering learner attributes, submitting the record, and viewing the resulting prediction. The interface should present the prediction together with enough context for an instructor to interpret it as a training-support signal.

Power BI is used to build a dashboard for student-performance trends and areas requiring improvement. Visualization can help instructors move from an individual prediction to a broader view of learner progress. At the individual level, the system can show a prediction and recommendations; at the school level, the dashboard can summarize trends across available records.

The components communicate through defined responsibilities. The frontend collects learner information and sends it to the FastAPI service. The service validates the request, invokes the preprocessing and prediction pipeline, and returns the result to the frontend. Database operations store learner records and



outcomes, while reporting tools can consume approved historical data for dashboard construction. This arrangement avoids embedding database or model logic directly inside the user interface.

The prediction request should be treated as a complete transaction. Before a model is called, the API can verify that required fields are present, that numerical values fall within reasonable ranges, and that categorical values belong to the permitted set. This validation reduces the chance that malformed inputs reach the model and also provides clearer feedback to the user. After validation, the same preprocessing object used during training should transform the record so that feature order, encoding, and scaling remain consistent.

Recommendations should be linked to the information that the system can actually observe. If the model uses practice hours, lessons, written-test score, and confidence, the recommendation layer can refer to these dimensions without claiming to diagnose a driving skill that was never represented in the data. For example, a lower readiness estimate can prompt an instructor to review practice exposure or written preparation. This keeps the recommendation mechanism aligned with the evidence available to the prediction system.

The system can also support historical comparison. When a learner completes additional lessons or practice sessions, a new record can be evaluated and the resulting readiness estimate can be compared with earlier predictions. Such a trend can help instructors observe whether the learner is moving toward readiness. However, comparisons should be interpreted carefully because changes in the input data, model version, or training context can affect the prediction.

An important implementation consideration is the treatment of new test results. After a learner completes the official test, the actual outcome can be stored with the original prediction context. This creates a historical example that can later be included in a controlled retraining dataset. The original input values should be preserved rather than replaced by later updates, because changing the original record can make it difficult to reconstruct what information was available when the prediction was produced.

The recommendation stage can be implemented as a separate rule-based layer above the model output. This separation is useful because the machine-learning model can focus on estimating the outcome while the recommendation logic translates the estimate into an instructional action. If the recommendation rules change, they can be updated without retraining the model. The rules should be documented and evaluated with instructors before being treated as part of the operational workflow.

The workflow should also support unsuccessful API calls and incomplete database operations. If a prediction cannot be generated, the user should receive a clear message and the system should avoid recording a misleading result. Similarly, if a learner record is stored but the prediction fails, the application should make the state of the transaction clear. These controls are especially important when the system is used as part of a live training process.

From a usability perspective, the number of required fields should be balanced against the quality of the prediction. Requiring many fields can increase data-entry effort and lead to incomplete records, while requiring too few fields may reduce predictive usefulness. The final interface should therefore distinguish mandatory fields from optional administrative information and should provide concise descriptions for fields such as confidence level and prior driving experience.

The project can also support role-specific access. Instructors may need individual learner predictions and recommendations, while administrators may primarily need aggregate trends and resource information. Although the source synopsis does not specify a role-management mechanism, the architecture can



accommodate this requirement through authenticated API access and controlled dashboard permissions during a future implementation.

Data validation should include checks on ranges and consistency. Practice hours and lesson counts should not be negative, written-test scores should remain within the defined scoring scale, and dates should be valid. Cross-field checks can also identify implausible combinations before a prediction is requested. Such validation improves both user experience and the quality of the historical dataset used for future model development.

FUNCTIONAL WORKFLOW

The end-to-end workflow can be described in seven steps. First, a learner record is entered into the system. Second, the record is stored in MySQL. Third, preprocessing cleans the input and transforms it into the representation expected by the model. Fourth, the trained model evaluates the processed attributes and returns a prediction. Fifth, FastAPI communicates the prediction to the application layer. Sixth, Streamlit presents the result to the user. Seventh, Power BI provides a broader analytical view while the recommendation stage can guide further practice or instruction.

The flowchart supplied with the project summarizes the process as Data Input, Data Cleaning, Model Training, Data Visualisation, Predict Results, Show Recommendations, and End. In implementation, visualization can operate alongside prediction as an analytical service. The key requirement is that each stage receives consistent inputs and produces outputs that the next stage can use.

Integration testing should verify the complete request path rather than testing only individual components. A valid learner record should reach the API, pass validation and preprocessing, generate a model response, and return a result that the Streamlit interface can display. Database persistence and retrieval should also be checked so that new outcomes can later contribute to model updates. Failures should produce clear messages rather than exposing internal implementation details.

DATA MODEL AND FEATURE PREPARATION

The learner record should be designed so that each field has one clear meaning and one expected representation. Date of birth can be stored as a date value, while derived age can be calculated at the appropriate point in the pipeline. Prior driving experience and vehicle type should use controlled categories rather than free-form text. Practice hours and lesson counts should be numeric, and written-test score should follow the scoring scale used by the driving school. Confidence should use a documented scale so that values entered by different users remain comparable.

The target variable should represent the outcome that the system is designed to predict. For a binary pass/fail task, the label should be derived from the official outcome recorded after the test. Records without a known outcome should not be treated as completed training examples merely because an instructor believes the learner was ready. Separating prediction inputs from the final label is important because the label is not available at the time a pre-test prediction is generated.

Feature preparation should also consider whether some attributes introduce unnecessary bias or administrative leakage. A learner name, database identifier, or other purely administrative value should not be included simply because it is available in the table. Similarly, information entered after the official test should not be used to predict the earlier outcome. The model input should contain only evidence that would reasonably be available when the prediction is requested.



Missing data require an explicit policy. A missing practice-hour value should not silently become zero because zero hours has a different meaning from an unknown value. Categorical fields should also distinguish an unrecorded value from a valid category when the application requires that distinction. These choices should be documented because preprocessing decisions can influence model behavior and reproducibility.

Storage design should distinguish current learner information from historical test outcomes. This makes it possible to preserve a record of what information was available at prediction time and what the eventual result was. A relational structure can maintain learner records, training events, prediction requests, and official outcomes as related entities. The exact database schema is not specified in the synopsis, so the implementation should define it according to the final application requirements.

The optional GPU requirement reflects the possibility that model training may become computationally demanding, but the project is based on structured learner data rather than image processing. Therefore, a CPU-based development environment may be sufficient for many candidate models. The hardware choice should follow the selected algorithm and dataset size rather than assuming that a GPU is mandatory.

Python provides the common environment connecting the data-processing and machine-learning components. Pandas can be used to transform tabular data, Matplotlib can support exploratory plots, and SQLAlchemy can manage database interaction. FastAPI separates service access from the prediction code, while Streamlit provides the user-facing workflow. Power BI then supplies a separate analytical environment for reporting.

IMPLEMENTATION REQUIREMENTS

The project can be implemented on a standard development workstation. The source specification requires an Intel i5 processor or higher, 8 GB RAM as a minimum with 16 GB recommended, approximately 5 GB storage, and an optional GPU for faster model training. The software environment includes Python 3.11+, Streamlit, Pandas, Matplotlib, SQLAlchemy, FastAPI, and Power BI.

TABLE I. PROJECT TECHNOLOGY AND HARDWARE REQUIREMENTS

Category	Requirement	Purpose
Processor	Intel i5 or higher	Application and model execution
Memory	8 GB minimum; 16 GB recommended	Data processing
Storage	5 GB	Data and application storage
GPU	Optional	Faster model training
Runtime	Python 3.11+	Development
Backend	FastAPI	API integration
Frontend	Streamlit	Web application
Database	MySQL	Persistent records
Visualization	Power BI	Dashboard



MODEL SELECTION AND VALIDATION

The synopsis does not prescribe a particular learning algorithm. A practical implementation can therefore begin with a small set of interpretable baseline models and then compare them with more complex alternatives. The baseline establishes whether the available attributes contain useful predictive information before additional model complexity is introduced. A more complex model should be adopted only when its validation behavior provides a meaningful benefit relative to computational cost and interpretability.

Training and validation should be separated from the final test evaluation. Hyperparameters such as regularization settings, tree depth, learning rate, or class weights should be selected using the training and validation process rather than by repeatedly examining the final test results. When the dataset is small, cross-validation can provide a more stable estimate of generalization, provided that the final test set remains isolated.

Class balance should be examined because a pass/fail dataset can become misleading when one outcome is much more common than the other. If most learners pass, a model that predicts pass for everyone may obtain a high accuracy while failing to identify learners who need support. Precision, recall, F1-score, confusion-matrix analysis, and class-specific error rates therefore provide more informative evidence for the intended use.

Probability outputs should be interpreted carefully. A predicted probability is not automatically a calibrated probability. If the application communicates values such as 0.80 or 80 percent likelihood, calibration should be checked on held-out data. The interface can then use clearly defined categories such as higher readiness, review recommended, or additional practice suggested, but the threshold for these categories should be established from validation evidence rather than selected arbitrarily.

Model stability is also relevant. Small changes in the training sample should not cause extreme changes in the prediction for similar learners. Repeated validation or resampling can help identify unstable behavior. If instability is observed, the development team should investigate data quantity, feature quality, class imbalance, or model complexity before deployment.

Evaluation can also examine threshold behavior because the practical use of a readiness prediction depends on how the output is converted into an action. A school may prefer to identify as many potentially unready learners as possible, even if this produces some false alarms. Another school may prefer fewer interventions and accept a higher risk of missing an at-risk learner. The threshold should therefore be selected with the intended training objective in mind and validated on historical data rather than chosen only for maximum accuracy.

Error analysis should be performed at the level of individual predictions as well as aggregate metrics. When a prediction differs from the eventual outcome, the development team can review the input record, preprocessing result, and model confidence to identify recurring patterns. For example, errors may cluster around learners with little prior experience, unusual vehicle-training combinations, or incomplete records. Such patterns can guide data-quality improvements and future feature design.

The dashboard can provide an important feedback loop for evaluation. Aggregated prediction outcomes can be compared with eventual test results after those results are available. This allows the school to determine whether the predictive signal is useful over time. The dashboard should clearly distinguish predicted values from official outcomes so that users do not confuse an estimate generated before the test with the final examination result.



Reproducibility should be considered throughout the evaluation process. The implementation should record the dataset period, preprocessing rules, selected features, model configuration, and evaluation protocol for each model version. If the system is later updated, the new version can then be compared with the previous version under a documented procedure. This makes performance changes easier to interpret and supports more reliable academic reporting.

EVALUATION STRATEGY

Since the source project does not report a completed experimental evaluation, the evaluation stage should be treated as a planned component of the implementation. The historical dataset should be divided into training and validation data, with a separate test portion reserved for final assessment when sufficient records are available. All preprocessing parameters must be learned from the training portion and then applied unchanged to validation and test records.

Evaluation should use measures appropriate to a binary pass/fail prediction. Accuracy can provide an overall view, but it should not be the only measure. Precision, recall, F1-score, and a confusion matrix can reveal whether the model is disproportionately missing learners who eventually pass or incorrectly labeling learners as ready. If the model produces probabilities, calibration should also be examined so that a reported likelihood has a meaningful interpretation.

Because the intended deployment is in driving schools, generalization is especially important. A model should be tested on data representing different learner groups, training patterns, and instructor practices when such data are available. Performance should be monitored after deployment, and new outcomes should be incorporated only through a controlled retraining process.

DASHBOARD AND USER INTERFACE DESIGN

The Streamlit interface should present the learner-input form in a sequence that matches the data-collection process. Required fields can be validated before the record is submitted to FastAPI. Numerical fields should reject impossible values, categorical fields should use predefined options, and the interface should clearly distinguish information required for prediction from information retained only for administration.

The prediction result should be displayed in a form that is understandable to a driving instructor. A single score without context may be misleading, so the interface can show the predicted outcome together with the main input information used for the request. Where recommendations are generated, they should be phrased as training-support suggestions rather than guarantees. For example, the system can recommend additional practice when the model indicates lower readiness, while the instructor remains responsible for deciding the actual training plan.

The Power BI dashboard should focus on aggregated information that supports planning and review. Possible views include the distribution of learner outcomes, practice-hour patterns, lesson attendance, written-test performance, and changes in predicted readiness over time, provided that these fields are present and consistently recorded. Individual-level information should not be exposed more broadly than necessary, particularly when dashboards are shared among multiple users.

The frontend, API, database, and dashboard should use consistent field names and definitions. A change in a database field should not silently alter the meaning of a prediction input. Versioning of the model and preprocessing pipeline can help maintain this consistency as the application evolves.



DATA MANAGEMENT AND RESPONSIBLE USE

The system stores learner information that can include names, dates of birth, training history, and self-reported confidence. Such information should be collected only for legitimate project purposes and protected through appropriate database access controls. Where possible, development datasets should use identifiers that are not required by the predictive model. Access to prediction results should also be limited to authorized users such as instructors or administrators.

A prediction can influence how a learner is perceived, so the output should be used as decision support rather than as an automatic judgment of competence. Instructors should retain the ability to review the underlying learner information and override a recommendation when professional observation provides additional evidence. Model updates should be documented so that changes in performance can be traced to changes in data, preprocessing, or the selected learning method.

TESTING AND DEPLOYMENT CONSIDERATIONS

Software testing should cover unit-level and integration-level behavior. Data-validation functions should be tested with valid, missing, boundary, and invalid inputs. API endpoints should be checked for expected request and response structures. Database operations should verify insertion, retrieval, and update behavior. The complete prediction path should then be tested from the Streamlit interface through FastAPI to the model and back to the user interface.

Deployment should separate development and production configurations. Database credentials and service configuration should not be embedded directly in application code. Access to the database should be limited to the operations required by the application, and prediction endpoints should be protected according to the deployment environment. Logging can help diagnose failures, but logs should avoid exposing unnecessary personal learner information.

Model deployment should also be versioned. Each deployed model can be associated with the preprocessing configuration, feature definitions, training period, and validation summary that produced it. If a later model behaves unexpectedly, this information allows the team to identify which version generated a particular prediction and to reproduce the deployment configuration.

Continuous monitoring is important because learner populations can change over time. A model trained on historical records may encounter different patterns as training practices, vehicle types, or learner demographics change. Monitoring should therefore examine both data characteristics and prediction outcomes. Retraining should occur through a documented process after sufficient new labeled outcomes have been collected.

Another limitation is the dependence on self-reported confidence. Confidence is subjective and can vary between learners even when their practical abilities are similar. The field can still be useful as one input, but it should not be interpreted as a direct measurement of competence. Consistent instructions for reporting confidence and validation against observed outcomes can help determine whether the feature contributes useful information.

Practice hours and lesson counts can also be measured differently across driving schools. One school may count supervised practice differently from another, or learners may report informal practice that is not recorded. These differences can reduce comparability. Standardized definitions and data-entry rules are therefore important if the model is later expanded to multiple schools.

The model may also reflect historical instructor practices. If historical records show that certain groups received different amounts of training before testing, a model may learn associations that are not causal.



The prediction should therefore be interpreted as an estimate based on historical patterns, not as proof that changing one variable will necessarily change the final test outcome.

Finally, the project synopsis does not describe a finalized user-authentication mechanism, database schema, model versioning procedure, or deployment infrastructure. These are implementation details that must be defined during system development. They should not be treated as completed capabilities until they have been implemented and tested.

LIMITATIONS

The project synopsis identifies three principal limitations. First, prediction accuracy depends heavily on the quality and quantity of training data. If historical records are sparse or inconsistent, the model may not learn stable relationships. Second, the model may not generalize well to driving schools that use different teaching methodologies. A model trained on one environment therefore requires additional validation before being used elsewhere.

Third, real-time integration with external systems, such as official driving-test result systems, is not currently supported. The current design depends on data entered and maintained within the application environment. This limitation affects automation and may require additional interfaces, authentication, data agreements, and validation if external systems are connected in the future.

FUTURE SCOPE

One proposed direction is to incorporate external information such as road conditions and weather data. Such variables could provide additional context when relevant and when reliable historical data are available. A future implementation could compare whether these additional features improve validation performance before adopting them.

A second direction is development of a mobile application for easier access and real-time updates. A mobile interface could allow instructors or learners to review training information outside the desktop environment while continuing to use the same backend services. The FastAPI layer could support such an extension without duplicating prediction logic in the mobile client.

Future development can also include a more formal model-monitoring process. Each retraining cycle can record the training-data period, preprocessing version, model version, validation measures, and deployment date. This would make it easier to determine whether an apparent improvement is due to a genuine change in predictive quality or a change in the learner population.

Another useful extension is to connect the prediction to individualized learning plans. If a learner has limited practical exposure, the application could highlight the need for additional practice; if written-test performance is weaker, the instructor could prioritize theoretical revision. Such recommendations should remain linked to available evidence and should be evaluated for usefulness rather than assumed to improve outcomes automatically.

CONCLUSION

The Driving Test Result Prediction System presents a machine-learning-based approach to estimating learner readiness for driving tests. The proposed solution combines learner attributes, data preprocessing, predictive modeling, API services, a web interface, and dashboard visualization into one integrated workflow. Its primary value is to provide instructors with a consistent, data-supported signal that can complement existing training and assessment practices.



The project emphasizes personalized guidance, improved resource allocation, and early identification of learners who may need additional support. At the same time, its limitations show that reliability depends on representative training data and appropriate validation across different teaching environments. Because the source synopsis does not provide measured experimental results or a finalized model, this paper intentionally does not claim a numerical accuracy or performance improvement. The proposed architecture provides a foundation on which those results can be established after implementation and evaluation.

REFERENCES

- [1] “Machine Learning with Python—A Hands-On Guide.”
- [2] FastAPI, “FastAPI Official Documentation.” [Online]. Available: <https://fastapi.tiangolo.com/>
- [3] MySQL, “MySQL Documentation.” [Online]. Available: <https://dev.mysql.com/doc/>
- [4] Microsoft, “Power BI Documentation.” [Online]. Available: <https://docs.microsoft.com/en-us/power-bi/>