

Grievance AI: Startup Ecosystem Intelligence Hub

Dhanyaa Shree T*, B. Meena Preethi**

*(Department of Software Systems and AIML, Sri Krishna Knowledge Foundation (Deemed to be University), Coimbatore-641008, Tamil Nadu, India

Email: dhanyaashreet23mss011@skasc.ac.in

** (Department of Software Systems and AIML, Sri Krishna Knowledge Foundation (Deemed to be University), Coimbatore-641008, Tamil Nadu, India

Email: meenapreethib@skasc.ac.in

Abstract:

Support organizations for early-stage startups receive grievances from founders about hiring, marketing, legal compliance, funding, and mentorship, and most of them still sort and assign these grievances by hand. This paper describes Grievance AI: Startup Ecosystem Intelligence Hub, a web platform that automates the first steps of that work. Grievance text is cleaned and converted into term frequency-inverse document frequency (TF-IDF) vectors, and a Support Vector Machine (SVM) or an XGBoost classifier assigns one of six categories. The predicted category selects the responsible officer, and the grievance then moves through a tracked lifecycle from submission to closure. Founders, officers, and administrators work through separate role-based interfaces. Founder accounts are created through a whitelist and verified with an email one-time password, and three service tiers (Standard, Elite, and Premium) control access to chat, document upload, and appointment-based calls. Founder ratings feed officer performance analytics. The system is built with Python, Flask, MySQL, and Scikit-learn. Its contribution is the integration of these parts into a single workflow, because the underlying algorithms are standard. The paper also defines the protocol for comparing the two classifiers using accuracy, precision, recall, and macro and weighted F1.

Keywords — Artificial Intelligence, Natural Language Processing, Grievance Management, TF-IDF, Support Vector Machine, XGBoost, Machine Learning, Intelligent Routing, Startup Ecosystem, Flask.

I. INTRODUCTION

A. Background and Motivation

Founders of young companies deal with hiring, marketing, legal obligations, fundraising, and the search for mentors, usually with very few people. Incubators, accelerators, and startup support cells put officers with the relevant experience at their disposal, but this help works only if a founder's problem reaches the right officer quickly. A structured grievance process also leaves a record that programme managers can audit and learn from. Progress in automatic text categorization [1], [2] makes it realistic to move the first triage step from staff to software.

B. Challenges in Existing Startup Grievance Management

In practice, grievances arrive by email, chat, or conversation. Someone must read each one and decide its category, assignment waits until that person is free, and a wrong guess sends the case to a handler who then forwards it, a hand-off problem that is well documented for IT service desks [3]. Founders cannot see what is happening to their request, replies are scattered over several channels, and it is hard to judge how individual officers perform or how many grievances of each kind arrive.

C. Research Gap

Text categorization [1], ticket routing [3], and access control [4] are each well studied on their own. We are not aware of a published framework that joins classification, category-based routing, a defined lifecycle, tiered support, founder feedback, and officer analytics for the startup ecosystem. Typical grievance portals stop at submission and tracking, and typical classifiers stop at predicting a label.

D. Problem Statement

A grievance is written as $G_i = (T_i, C_i)$, where T_i is the free text and C_i belongs to the label set $\mathcal{C} = \{\text{Hiring, Marketing, Legal Compliance, Funding, Mentorship, General}\}$. The problem is to learn a mapping f from text to $\mathcal{C}$ using labeled grievances, to compose it with a routing function ρ from $\mathcal{C}$ to the set of officers $\mathcal{O}$, and to manage the resulting assignment through a lifecycle whose recorded events support feedback and performance analytics, all inside an authenticated, role-restricted platform.

E. Objectives

The platform is meant to (1) classify grievances automatically with NLP preprocessing, TF-IDF features, and SVM and XGBoost classifiers; (2) route each grievance to the responsible officer; (3) restrict access through whitelist registration, email OTP verification, and role-based control; (4) offer Standard, Elite, and Premium support tiers; (5) track every grievance through a fixed lifecycle with founder feedback; and (6) give administrators officer-level analytics.

F. Contributions and Novelty

The main contribution is a single workflow that links NLP, TF-IDF, ML classification, routing, resolution, feedback, and analytics. In detail, the paper offers (i) an end-to-end grievance pipeline embedded in a role-based web platform, (ii) a protocol for comparing SVM and XGBoost on shared TF-IDF features, (iii) a category-based routing function tied to a formal status-transition model, and (iv) a tier-aware service layer whose founder feedback drives officer analytics. TF-IDF, SVM, and XGBoost are established methods, so novelty is claimed only for their integration in this setting.

II. RELATED WORK

A. Conventional Grievance Management Systems

Conventional grievance and help-desk platforms rely on form submission, manual triage, and status tracking, so their quality depends on human classification. Work on service-desk ticket routing shows that mining historical resolution sequences can cut unnecessary transfers [3], which motivates automating the categorization step.

B. NLP-Based Text Classification

Supervised text categorization assigns documents to predefined classes. Its pipeline, with tokenization, normalization, feature construction, and a classifier, is surveyed in [1] and more recently in [2]. Comparative studies on benchmark corpora examine which classifiers work best when classes differ in size [5].

C. TF-IDF-Based Feature Engineering

TF-IDF weights a term by its frequency in a document and by its rarity in the collection. The inverse-document-frequency idea comes from the statistical interpretation of term specificity [6], term-weighting variants were compared systematically in [7], and [8] gives a textbook treatment. The result is a sparse vector that suits short, domain-specific text.

D. SVM for Text Classification

The support-vector network [9] finds a maximum-margin hyperplane. Joachims [10] observed that text tasks have many features, most of them relevant, and are often nearly linearly separable, which suits SVMs. Yang and Liu [5] found SVM and kNN stronger than Naive Bayes and neural-network baselines when categories had few positive examples. Multi-class strategies are compared in [11].

E. XGBoost for Classification

Gradient boosting builds an additive ensemble by fitting each new learner to the gradient of the loss [12]. XGBoost [13] adds a regularized objective, second-order optimization, and sparsity-aware split finding, so it can be applied to sparse TF-IDF matrices; see also the official documentation [14].

F. Intelligent Ticket and Complaint Routing

Routing research aims to deliver each request to a competent handler with few hand-offs [3]. Classification-driven routing, where the predicted class selects the responsible group, is the simplest form and is the one used here. Policies that consider workload and expertise are left for future work.

G. Research Gap

Table I compares categories of approaches at the capability level; it does not evaluate particular products. The proposed system differs in covering all listed capabilities within one platform.

TABLE I
COMPARISON OF EXISTING APPROACHES AND THE PROPOSED SYSTEM

Capability	Manual	Rules	ML-only	Proposed
Manual sorting	Y	P	N	N
NLP processing	N	N	Y	Y
TF-IDF	N	N	P	Y
Machine learning	N	N	Y	Y
Auto routing	N	Y	P	Y
Role-based access	P	Y	N	Y
Tracking	P	Y	N	Y
Feedback	P	P	N	Y
Analytics	N	P	N	Y
Service tiers	N	N	N	Y
Appointments	N	N	N	Y

Manual: manual or portal-based; Rules: rule-based ticketing; ML-only: stand-alone classifier. Y: typically supported; P: partial; N: not typically supported.

III. PROPOSED GRIEVANCE AI FRAMEWORK

A. System Overview

The framework is a web platform that accepts grievances from authenticated founders, classifies them with a trained model, routes them to category-specific officers, and records the resolution, feedback, and performance data. Founders, officers, and administrators use separate interfaces.

B. Design Objectives

Five goals guided the design: automate first-level triage, separate the duties of the three roles, keep a trace of every grievance event, keep the classifier replaceable without touching workflow logic, and limit services by founder tier.

C. Multi-Layer Architecture

As shown in Fig. 1, requests from the three interfaces pass through the authentication layer to the Flask application layer. Grievance text then

goes through the AI processing pipeline (NLP processing, TF-IDF extraction, ML classification, and routing). Routing outcomes are stored in MySQL, and the analytics component reads from the same database.

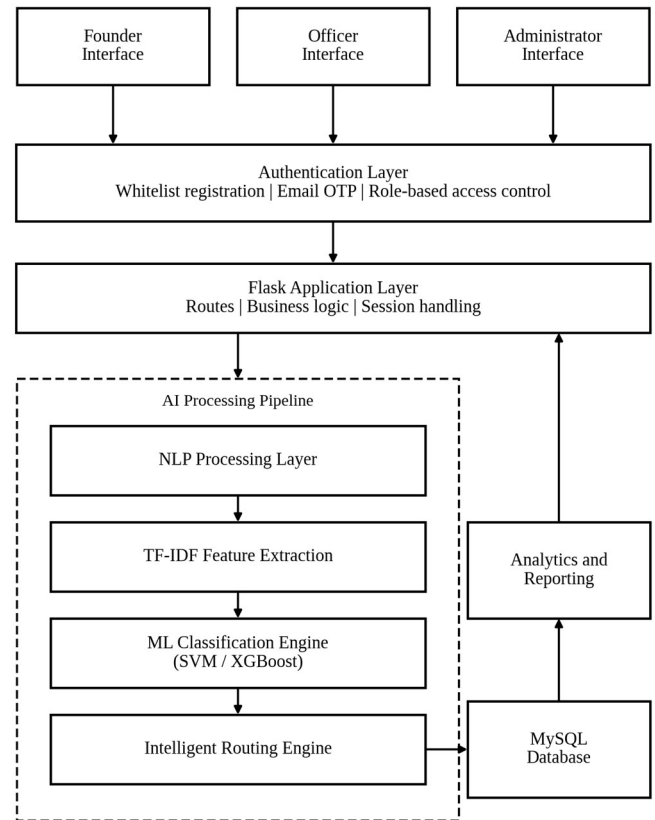


Fig. 1. Overall Architecture of the Proposed Grievance AI Framework

D. AI-Driven Grievance Processing

A submitted grievance is cleaned, tokenized, vectorized with a vocabulary fitted on training data, and passed to the selected classifier, which returns a category label (Section IV).

E. Intelligent Category-to-Officer Mapping

Every category is tied to a responsible officer role. The routing engine applies this mapping to the predicted label and creates the assignment record (Section V-B).

F. Tier-Aware Support Framework

Each founder belongs to the Standard, Elite, or Premium tier, and the tier decides which support channels are available (Section VI-E).

G. End-to-End Grievance Lifecycle

A founder registers, authenticates, and submits a grievance. The text is preprocessed, vectorized, classified, and mapped to an officer, who works on it and records a resolution. The founder then gives feedback, and the stored events feed performance analytics. Fig. 3 shows the routing and resolution part of this flow.

IV. GRIEVANCE CLASSIFICATION METHODOLOGY

The pipeline in Fig. 2 turns raw grievance text into a category label through normalization, TF-IDF vectorization, and one of two candidate classifiers.

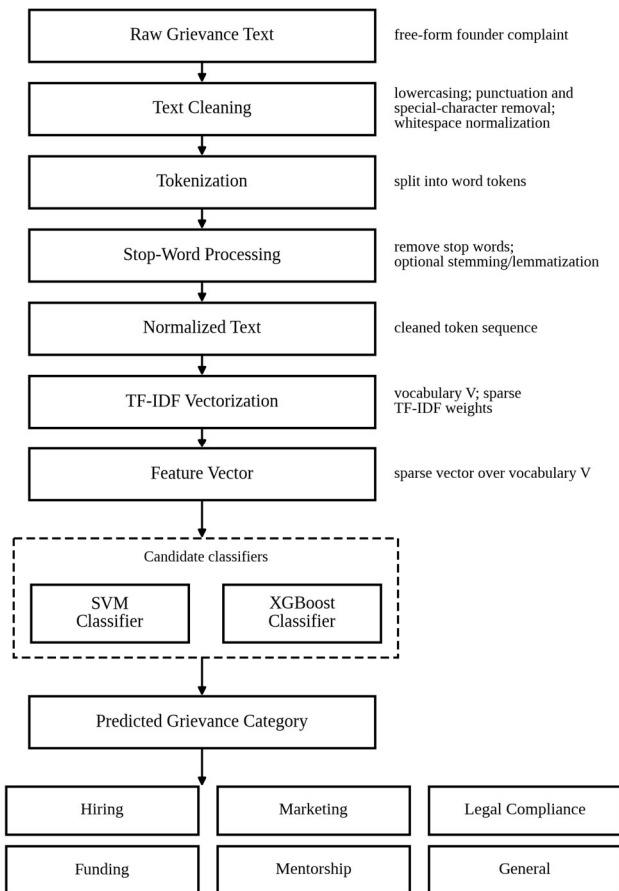


Fig. 2. NLP and Machine Learning-Based Grievance Classification Pipeline

A. Grievance Dataset Construction

Each labeled grievance is the pair $G_i = (T_i, C_i)$ defined in Section I-D. The labels are Hiring, Marketing, Legal Compliance, Funding, Mentorship, and General, where General collects grievances that fit none of the specialized categories. Category frequencies are rarely equal in practice,

and imbalance affects how results should be read [15], so the data are split with stratification to keep class proportions in both partitions. The size and class counts of each corpus version are recorded with the training run.

B. Text Preprocessing

Preprocessing lowercases the text, removes punctuation and special characters, tokenizes, removes stop words, and normalizes whitespace; stemming or lemmatization can be added to merge inflected forms. For example, “Our investor’s term-sheet clause is unclear, can legal help??” becomes the tokens investor, term, sheet, clause, unclear, legal, help once a standard English stop-word list is applied. The terms investor and legal in this example also show why Funding and Legal Compliance can overlap.

C. TF-IDF Feature Engineering

Let $f(t,d)$ be the count of term t in document d . The term frequency is

$$TF(t,d) = f(t,d) / \sum_k f(k,d) \quad (1)$$

and, for a collection of N documents of which $df(t)$ contain t , the inverse document frequency is

$$IDF(t) = \log(N / df(t)) \quad (2)$$

so that the weight of a term in a document is

$$TFIDF(t,d) = TF(t,d) \times IDF(t) \quad (3)$$

Each grievance becomes a sparse vector x_i over the vocabulary V . Short grievances give sparse vectors in which category-specific words carry high weights and common words are suppressed [7], [8], which is why TF-IDF fits this task. Scikit-learn implements a smoothed IDF with L2 normalization [16]. The vocabulary and IDF values are fitted on the training partition only, so no information from the test data leaks into the features.

D. SVM-Based Classification

An SVM works in the high-dimensional TF-IDF space and looks for the hyperplane with the largest margin between classes [9]. The soft-margin problem is

$$\min_{w,b,\xi} \frac{1}{2} \|w\|^2 + C \sum_i \xi_i \quad (4)$$

subject to $y_i(w^T x_i + b) \geq 1 - \xi_i$ and $\xi_i \geq 0$ for the n training samples, where C balances margin width against training error. The samples on or inside the margin are the support vectors, and only they

determine the hyperplane. With $K = |\mathcal{C}|$ categories, a one-versus-rest scheme trains one scoring function per class,

$$f_k(x) = \langle w_k, x \rangle + b_k \quad (5)$$

and the predicted class is the one with the highest score:

$$\hat{y} = \operatorname{argmax}_k f_k(x) \quad (6)$$

One-versus-one decomposition is the alternative [11]. We use a linear kernel, which is the usual choice for text [10], with $C = 1.0$ and balanced class weights.

E. XGBoost-Based Classification

Gradient boosting grows a decision-tree ensemble in which every new tree corrects the errors of the trees before it [12]. The prediction for sample i is the sum of K tree outputs,

$$\hat{y}_i = \sum_k f_k(x_i), \quad f_k \in \mathcal{F} \quad (7)$$

and training minimizes the regularized objective

$$L = \sum_i l(y_i, \hat{y}_i) + \sum_k \Omega(f_k) \quad (8)$$

where l is a differentiable loss (multi-class softmax here) and the penalty

$$\Omega(f) = \gamma T + \frac{1}{2} \lambda \sum_j (w_j)^2 \quad (9)$$

limits tree complexity through the number of leaves T and the leaf weights w_j [13]. Sequential fitting, shrinkage, and regularization let the model capture non-linear term interactions, and sparsity-aware splitting handles sparse input. The configuration is 200 trees of maximum depth 6, a learning rate of 0.1, and the multi-class softmax objective.

F. Comparative Model Selection

Both classifiers are trained on identical TF-IDF features and evaluated on the same held-out partition. They are compared on accuracy, precision, recall, F1-score, macro F1, weighted F1, and training time. Since class imbalance can make accuracy misleading [15], [17], macro F1 is the main criterion, and training time breaks ties. The deployed model is the one that scores higher under this rule.

V. OFFICER ROUTING AND GRIEVANCE LIFECYCLE

A. Category-Aware Officer Assignment

Each category corresponds to an officer role, so a grievance reaches someone with relevant expertise

without manual triage. The assignment is created automatically after classification and stored in the database.

B. Intelligent Routing Mechanism

Routing is a mapping ρ from $\mathcal{C}$ to $\mathcal{O}$, so the officer for grievance i with predicted category $\hat{c}_i$ is

$$o_i = \rho(\hat{c}_i) \quad (10)$$

The mapping is Hiring to HR Officer, Marketing to Marketing Officer, Legal Compliance to Legal Officer, Funding to Funding Officer, Mentorship to Mentorship Officer, and General to General Support Officer. This is a category-to-officer lookup only. It does not balance workload, check availability, or optimize anything; the learned classifier that produces $\hat{c}_i$ is the intelligent part.

C. Grievance Status Transition Model

The lifecycle in Fig. 3 is a forward-only state machine over Submitted, Assigned, In Progress, Resolved, and Closed. A grievance is Submitted on creation, becomes Assigned when routing creates the assignment, In Progress when the officer starts work, Resolved when the officer records a resolution, and Closed once founder feedback completes the lifecycle. Transitions outside this order are rejected by the application, and each accepted transition is stored with a timestamp, which gives the event history used for analytics.

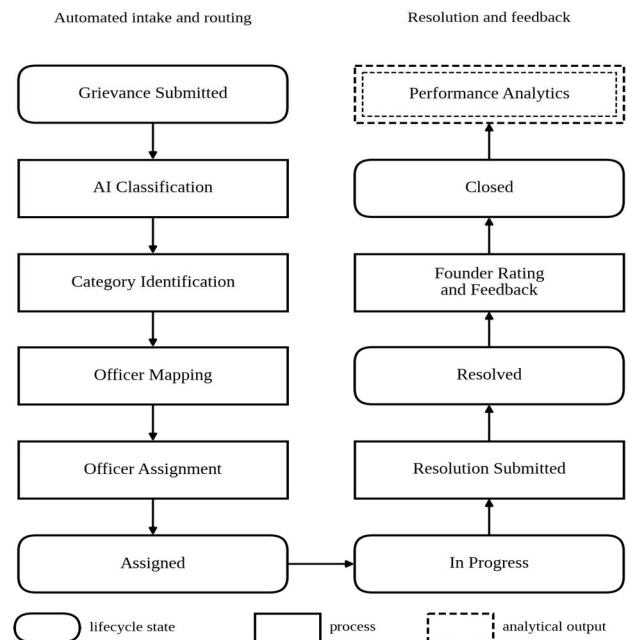


Fig. 3. Intelligent Grievance Routing and Resolution Lifecycle

D. Resolution Management

The assigned officer records the outcome against the grievance and updates its status. The founder can check the status and resolution on the dashboard, which gives the visibility that informal channels lack.

E. Founder Feedback and Officer Rating

After resolution the founder rates the handling officer on a scale of 1 to 5 and may add a comment. The rating is linked to both the grievance and the officer.

F. Performance Analytics

For every officer o with handled-grievance set G_o , the platform reports the number of grievances handled, pending, and resolved, the mean resolution time, and the mean founder rating:

$$\bar{T}_o = (1 / |G_o|) \sum_g [t_{res}(g) - t_{sub}(g)] \quad (11)$$

$$\bar{R}_o = (1 / |G_o|) \sum_g r(g) \quad (12)$$

Here the sums run over g in G_o , t_{res} and t_{sub} are the resolution and submission times, and $r(g)$ is the rating, each taken over the resolved or rated subset. Administrators see these values per officer and per category.

VI. ROLE-BASED ARCHITECTURE AND TIER-AWARE SERVICES

Table II lists the modules, and Fig. 4 shows the role and tier structure.

TABLE II
SYSTEM MODULES AND FUNCTIONAL RESPONSIBILITIES

Module	Responsibilities
Founder	Dashboard, submission, tracking, history, services, feedback
Officer	Assigned grievances, status updates, resolutions, reports
Administrator	Founders, officers, categories, monitoring, analytics, reports
Authentication	Whitelist, email OTP, role-based access, tiers, appointments
NLP/ML engine	Preprocessing, TF-IDF, SVM/XGBoost prediction
Routing	Officer mapping, assignment, status transitions

A. Founder Module

The founder module offers a dashboard, grievance submission, status tracking, grievance history, selection of the services available for the founder's tier, and feedback after resolution.

B. Officer Module

Officers see the grievances assigned to them, update status, record resolutions, and view reports on their own cases.

C. Administrator Module

Administrators manage founder and officer accounts and grievance categories, monitor grievances across the platform, review analytics, and generate reports.

D. Authentication and Authorization

Registration is limited by a whitelist, so only founders approved by the organization can create accounts, and an email OTP verifies the address. Authorization follows role-based access control [4]: each account has one role, and that role determines which routes and data it can reach. Founders therefore cannot open officer or administrator functions, and officers cannot open administrative ones.

E. Tier-Based Support Services

Fig. 4 shows the three tiers. Standard gives chat support, Elite adds file and document upload, and Premium adds appointment-based call support. The tiers are cumulative, and a founder's tier decides which options are shown.

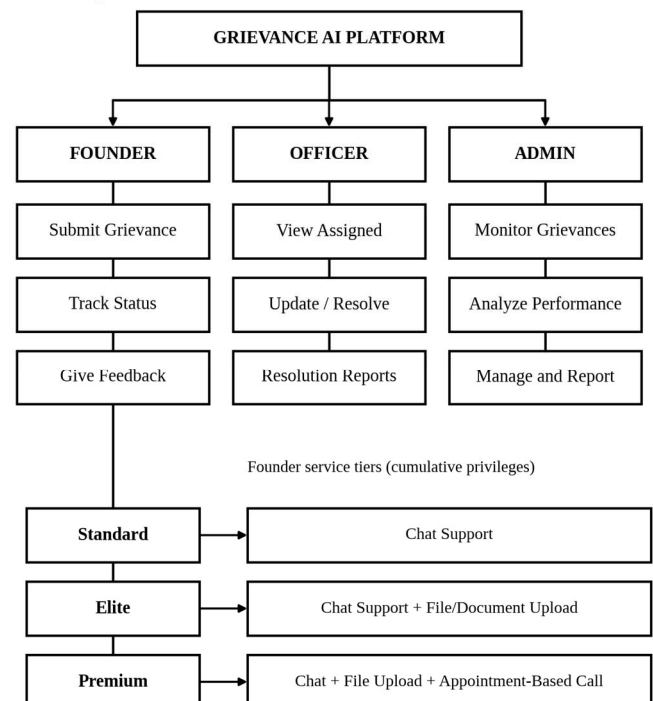


Fig. 4. Role-Based and Tier-Aware Service Architecture

F. Appointment Management

Premium founders book calls by choosing from the available time slots. A booking stores the founder and the chosen slot, and a booked slot is no longer offered to others.

VII. SYSTEM IMPLEMENTATION

A. Technology Stack

The backend is written in Python with the Flask framework. The interface uses HTML, CSS, JavaScript, and Bootstrap, and MySQL provides relational storage. Scikit-learn [16] supplies text vectorization and the SVM, XGBoost [13] supplies the boosted-tree classifier, and Pandas and NumPy handle data preparation and numerical work.

B. Backend Architecture

Flask routes receive requests from the three interfaces and pass them to application logic for authentication, grievance processing, status updates, feedback, appointments, and reporting. That logic talks to MySQL for persistent state and to the ML component for category prediction.

C. Machine Learning Integration

The fitted vectorizer and the selected classifier are trained offline and loaded by the Flask application. A new grievance receives the same preprocessing as the training data, is vectorized with the stored vocabulary, and is classified, and the label goes to the routing function in Eq. (10). Using one preprocessing definition for training and serving avoids mismatches between the two.

D. Database Design

The relational schema covers users, founders, officers, categories, grievances, assignments, status updates, service tiers, appointments, and feedback and ratings. A grievance references a founder and a category, an assignment links a grievance to an officer, status updates hold timestamped transitions, and feedback links a rating to a grievance and an officer.

E. Security Considerations

Security rests on whitelist registration, email OTP verification, role-based authorization on every protected route, database access only through the

application layer, and validation of user input. No other security mechanism is claimed.

VIII. EVALUATION METHODOLOGY AND ANALYSIS

A. Experimental Setup

Table III gives the evaluation configuration used for comparing the two classifiers.

TABLE III
EVALUATION CONFIGURATION

Item	Setting
Data split	Stratified 80/20 train/test
Validation	5-fold stratified cross-validation on the training part
Features	TF-IDF, unigrams and bigrams, minimum document frequency 2
SVM	Linear kernel, $C = 1.0$, one-versus-rest, balanced class weights
XGBoost	200 trees, depth 6, learning rate 0.1, softmax objective
Selection	Higher macro F1; shorter training time on ties

B. Dataset and Class Balance

The corpus holds one record per grievance with one of the six labels. Per-class counts and percentages are reported with every run, and the stratified split keeps them equal in both partitions.

C. Evaluation Metrics

For n test samples, accuracy is

$$\text{Acc} = (1/n) \sum_i I(\hat{y}_i = y_i) \quad (13)$$

For class c , with true positives TP_c , false positives FP_c , and false negatives FN_c , precision, recall, and F1-score are

$$P_c = TP_c / (TP_c + FP_c) \quad (14)$$

$$R_c = TP_c / (TP_c + FN_c) \quad (15)$$

$$F1_c = 2 P_c R_c / (P_c + R_c) \quad (16)$$

Macro F1 averages the class scores equally, and weighted F1 weights each class by its support n_c [17]:

$$\text{Macro-F1} = (1/|\mathcal{C}|) \sum_c F1_c \quad (17)$$

$$\text{Weighted-F1} = \sum_c (n_c/n) F1_c \quad (18)$$

D. Classifier Comparison Procedure

For each classifier the evaluation reports overall accuracy, macro-averaged precision, recall, and F1, weighted F1, per-category precision, recall, and F1, the confusion matrix, and wall-clock training time. A supplied evaluation script produces these values from any labeled corpus using the configuration in

Table III. This paper specifies the protocol and does not state performance figures, because they depend on the corpus used.

E. Confusion Matrix Analysis

Some category pairs share vocabulary and are the likeliest to be confused. Funding and Legal Compliance overlap in investment agreements and regulatory terms, Hiring and Legal Compliance in employment contracts, and General overlaps with every specialized category because its content is heterogeneous. The confusion matrix produced by the evaluation script is the place to confirm or reject these expectations.

F. Routing Efficiency

Automatic classification removes the manual categorization step and makes assignment immediate on submission. Since routing is a deterministic function of the predicted label (Eq. (10)), the correctness of routing equals the correctness of classification. Measuring the actual reduction in triage delay and misrouting requires deployment data.

G. Discussion

Results from the protocol should be read through macro F1 and per-category scores rather than accuracy alone, and a model that wins on macro F1 but is much slower to train should be weighed accordingly.

IX. NOVELTY, PRACTICAL SIGNIFICANCE, AND LIMITATIONS

A. Technical Novelty

The novelty lies in the architecture and workflow, not in the algorithms: TF-IDF, SVM, XGBoost, and Flask are established tools. What is new is combining NLP-based grievance understanding, TF-IDF features, comparative SVM and XGBoost classification, category-based routing, lifecycle tracking, tier-aware support, founder feedback, and officer analytics in one operational platform for the startup ecosystem.

B. Practical Significance

Incubators, accelerators, startup support cells, and innovation centers can use the framework to reduce manual triage, show founders where their grievance

stands, and give administrators evidence about officers and programme quality.

C. Limitations

Classification quality depends on the amount, variety, and labeling consistency of historical data. Categories are fixed in advance and can blur at their boundaries, and short or poorly written grievances give TF-IDF models little to work with because they do not capture deeper semantics. Routing depends only on the predicted category, and multilingual input is not handled. These are directions for further research.

X. FUTURE ENHANCEMENTS

Transformer models such as BERT [18] could capture context beyond TF-IDF. Support for Indian and regional languages would widen access. Sentiment and urgency detection could flag emotional or time-critical cases, and priority prediction could order each officer's queue. Explanation methods such as LIME or SHAP [19], [20] could show which terms drove a predicted category. Officer selection could weigh workload, availability, expertise, and past performance instead of using a fixed mapping. A conversational assistant could give preliminary guidance before submission, and predictive analytics could forecast grievance volume, resolution time, and escalation risk.

XI. CONCLUSION

This paper presented Grievance AI, an integrated framework for startup grievance management. NLP preprocessing and TF-IDF features feed SVM and XGBoost classifiers, and the predicted category routes each grievance to the responsible officer. A status-transition lifecycle, role-based access with whitelist registration and email OTP verification, tier-aware services, and founder feedback complete the platform and yield officer performance analytics. The contribution is the integration for incubators and startup support cells. Applying the evaluation protocol to a labeled corpus and building the extensions described in Section X are the next steps.

REFERENCES

- [1] F. Sebastiani, "Machine learning in automated text categorization," *ACM Comput. Surv.*, vol. 34, no. 1, pp. 1–47, Mar. 2002.

- [2] K. Kowsari, K. Jafari Meimandi, M. Heidarysafa, S. Mendu, L. Barnes, and D. Brown, "Text classification algorithms: A survey," *Information*, vol. 10, no. 4, Art. no. 150, 2019.
- [3] Q. Shao, Y. Chen, S. Tao, X. Yan, and N. Anerousis, "Efficient ticket routing by resolution sequence mining," in *Proc. 14th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2008, pp. 605–613.
- [4] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman, "Role-based access control models," *Computer*, vol. 29, no. 2, pp. 38–47, Feb. 1996.
- [5] Y. Yang and X. Liu, "A re-examination of text categorization methods," in *Proc. 22nd Annu. Int. ACM SIGIR Conf. Res. Develop. Inf. Retrieval*, 1999, pp. 42–49.
- [6] K. Spärck Jones, "A statistical interpretation of term specificity and its application in retrieval," *J. Documentation*, vol. 28, no. 1, pp. 11–21, 1972.
- [7] G. Salton and C. Buckley, "Term-weighting approaches in automatic text retrieval," *Inf. Process. Manage.*, vol. 24, no. 5, pp. 513–523, 1988.
- [8] C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to Information Retrieval*. Cambridge, U.K.: Cambridge Univ. Press, 2008.
- [9] C. Cortes and V. Vapnik, "Support-vector networks," *Mach. Learn.*, vol. 20, no. 3, pp. 273–297, 1995.
- [10] T. Joachims, "Text categorization with support vector machines: Learning with many relevant features," in *Proc. Eur. Conf. Mach. Learn.*, 1998, pp. 137–142.
- [11] C.-W. Hsu and C.-J. Lin, "A comparison of methods for multiclass support vector machines," *IEEE Trans. Neural Netw.*, vol. 13, no. 2, pp. 415–425, Mar. 2002.
- [12] J. H. Friedman, "Greedy function approximation: A gradient boosting machine," *Ann. Statist.*, vol. 29, no. 5, pp. 1189–1232, 2001.
- [13] T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," in *Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2016, pp. 785–794.
- [14] XGBoost Developers, "XGBoost documentation." [Online]. Available: <https://xgboost.readthedocs.io>
- [15] H. He and E. A. Garcia, "Learning from imbalanced data," *IEEE Trans. Knowl. Data Eng.*, vol. 21, no. 9, pp. 1263–1284, Sep. 2009.
- [16] F. Pedregosa et al., "Scikit-learn: Machine learning in Python," *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, 2011.
- [17] M. Sokolova and G. Lapalme, "A systematic analysis of performance measures for classification tasks," *Inf. Process. Manage.*, vol. 45, no. 4, pp. 427–437, 2009.
- [18] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *Proc. NAACL-HLT*, 2019, pp. 4171–4186.
- [19] M. T. Ribeiro, S. Singh, and C. Guestrin, "'Why should I trust you?': Explaining the predictions of any classifier," in *Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2016, pp. 1135–1144.
- [20] S. M. Lundberg and S.-I. Lee, "A unified approach to interpreting model predictions," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 30, 2017, pp. 4765–4774.