ChatGPT

Introduction and Business Alignment

The pipeline under review targets **early detection of quality defects** in statin tablet manufacturing by forecasting process sensor trends and predicting downstream quality failures. This aligns well with the business need: statin tablet plants incur ≈\$1.2M/year from batches failing final tests [1]. By forecasting 30–60 minutes ahead and flagging a "yellow" alert when defect risk exceeds 0.7, the system aims to catch adverse trends long before a batch is scrapped. In principle, using real-time sensor data (e.g. compression force, tablet speed, fill depth, force variability) to anticipate problems is appropriate for direct-compression tablet production [2] [3]. However, ultimate effectiveness depends on model accuracy and alignment with pharmaceutical risk tolerance: **missing a defect is very costly**, while excessive false alarms (which slow production and increase sampling) are also undesirable.

Data and Pipeline Overview

The dataset comprises 1,005 batches of cholesterol-lowering tablets (four strengths, nine batch sizes) with high-resolution sensor logs (16 sensors at 10-second intervals [4]) and batch-level quality results (drug dissolution, impurities, solvents). The two-stage pipeline is: (1) an LSTM encoder–decoder forecasting selected sensors over the next 30 min (given the last 60 min); (2) a classification model (XGBoost) using *current* and *forecasted* sensor statistics to predict the batch's defect probability. A defect is defined by failing final criteria (e.g. drug release <90% or impurity/solvent limits breached) [5]. A "yellow flag" alert triggers if $P(\text{defect}) > 0.7$. This design ostensibly provides up to 30 min lead time for intervention.

Key alignment points: the sensors chosen (main compression force, press speed, force variability *SREL*, fill depth) are critical process parameters in tablet compression [6] [7]. Forecasting them may anticipate drifts (e.g. a rising SREL variance may pre-empt capping problems). The chronological split used (70% train, 15% val, 15% test) preserves time order, avoiding lookahead bias [8]. The pipeline's goal – early warning well before batch completion – is conceptually sound for reducing out-of-spec batches.

Stage 1 – LSTM Forecasting

The LSTM model uses a 60 min input (360 timesteps) to predict 30 min ahead (180 timesteps) for four sensors. The chosen **encoder–decoder with RepeatVector** architecture (three LSTM layers encoding, two decoding) is reasonable for multi-step forecasts [9] [10]. Training produced roughly 144K overlapping sequences, of which ~32% were defect-associated [11].

Forecast accuracy: On test data the averaged errors were:

- **Compression force (main_comp):** RMSE ≈ 1.18 kN, MAE ≈ 0.59 kN [12]. If typical force is ~15 kN, this is ~4–8% error.
- **Tablet speed:** RMSE ≈ 35.8 tablets/hr, MAE ≈ 23.8 [12]. Given presses run ~100–180k tablets/hr [13], error is negligible (<0.1%).

- **Force variance (SREL):** RMSE ≈ 5.32 (units), MAE ≈ 1.99 [14]. Relative error depends on SREL's range (e.g. if SREL ~20, ~26% error).
- **Fill depth:** RMSE ≈ 0.66 mm, MAE ≈ 0.35 mm [15]. Typical fill depths are a few mm, so this may be ~15–25% error.

These error levels are mixed. The speed forecast is extremely accurate (so not limiting classification), but **fill and SREL forecasts have substantial uncertainty**. In pharmaceutical context, 20–30% forecast error in a critical sensor could degrade early-warning reliability. The use of MSE/MAE is standard, but it would also be helpful to evaluate *relative* or *percent* errors and distribution of errors; the code did not report MAPE or any normalized metric. In a regulated environment, one would also want to check that forecasting errors stay within control limits of process variability (for instance, is 0.66 mm fill error within specification tolerance?).

Furthermore, forecasting can propagate uncertainty: if the LSTM is imperfect, classification accuracy suffers. One potential improvement is to quantify forecast confidence (e.g. prediction intervals) or to train the LSTM to explicitly minimize classification loss (multi-task learning: forecast + defect). Alternatively, a direct classification model (e.g. an LSTM over the recent 60 min to predict defect) could bypass forecasting entirely, reducing one source of error. However, forecasting can exploit longer-term trends and may catch anomalies that static snapshots miss.

Pharma standards: There are no hard numerical "must-pass" thresholds for these metrics, but high accuracy is expected, especially for parameters tied to quality. In practice, control charts or statistical process control (SPC) often use 3-sigma limits; here the RMSE on fill (0.66 mm) should be compared to normal variation. If typical fill std is ~0.5 mm, RMSE > 1×std may be marginal. The model's training curves (not shown) and use of early stopping are good practices, but additional validation under different operating conditions (e.g. different batch sizes, speeds) would be needed to ensure robustness.

Stage 2 – Defect Classification

The classification uses 51 features derived from the **last 30 min of current sensor data** and the **30 min forecasted trajectory** of each sensor (mean, std, max, min, trend, "drift" from first to last 10 min) [16] [17]. It also includes simple cross-sensor ratios (e.g. speed-to-force) and time-in-batch. This is a rich feature set, but several points stand out:

- **Feature selection:** The model identified "SREL_forecast_min" and "tbl_fill_current_max" among top predictors [18]. This suggests low forecasted SREL values (perhaps indicating steady force) and recent high fill depths correlate with defects. However, many features are highly correlated (mean vs max vs last value). Dimensionality reduction or feature pruning could simplify the model and aid interpretability — important in a regulated setting.

- **Model choice:** The code trained an XGBoost classifier with class-weight adjustment (`scale_pos_weight`) and a separate GradientBoosting (sklearn) for comparison [19] [20]. The XGBoost hyperparameters (depth 6, 200 trees, 0.1 learning rate) are reasonable defaults. It uses early stopping on validation AUC, which is good practice. However, **the XGBoost performance at the operational threshold was effectively zero**: at threshold 0.7 it predicted *no* defect positives (precision=0, recall=0) [21]. In contrast, the simpler GradientBoosting yielded AUC ≈0.751 with precision 0.576, recall 0.642, F1 ≈0.607 and specificity 0.898 [22].

- **Thresholding issue:** A defect probability threshold of 0.7 was arbitrarily chosen. The XGBoost model's predictions must have been mostly below 0.7, so no flags were raised (TN=17821, FP=0, FN=3851, TP=0 [23]). This is a fatal operational flaw: the alert system would never catch any defect. Even for the better model (GradientBoosting), at 0.7 many actual defects were missed (recall ~64%). By pharma safety standards, missing ~36% of defects is unacceptable. False negatives (escaped defects) are usually costlier than false positives.

- **Imbalance handling:** The dataset's defect rate (~32% in sequences [24]) is moderate, but the split led to 3851 defects in test vs 17821 non-defects [25], i.e. ~18% positive. The XGBoost used a static weight but still failed on threshold. It may be **calibrated** (predicted probabilities might be too low). The GradientBoosting with no explicit class weighting performed better, interestingly. Both models should be further tuned: for XGBoost, consider adjusting `scale_pos_weight`, using `max_delta_step`, or trying a smaller threshold. For any model, precision–recall curves should guide threshold choice.

- **Evaluation metrics:** The report displays AUC, precision, recall, F1, specificity [26]. While AUC ~0.74 indicates some discriminative power, the absolute values of recall/precision must meet industry needs. In pharma, a low recall (many missed defects) is not acceptable. Depending on risk tolerance, one might require >90% sensitivity. Metrics should be benchmarked to historical SPC performance (e.g. current 10% sampling yields 72% defect detection [27]). The project doc suggests aiming for ~94% detection with ML [27] – far above the current ~0–64%. Also, reporting only threshold-level metrics is limiting; the precision–recall curve (which the code plots) should be analyzed to pick a threshold that balances false alarms vs missed defects according to cost.

Mismatches and Limitations

Several issues reduce the pipeline's real-world reliability:

- **Threshold too high:** As noted, the static 0.7 cutoff is not justified by analysis. It effectively silenced XGBoost's alerts. The model outputs should be calibrated and the alarm threshold tuned (perhaps via cost-based analysis).

- **Single threshold for all batches:** Different products or batch sizes may have different risk profiles. A one-size threshold ignores context. Adaptive or batch-specific thresholds (based on prior risk, product specification) might improve performance.

- **Model validation:** The report shows one train/val/test split. Time-series data often require *cross-validation over batches* or multiple rollings to ensure stability. Also, the absence of unseen-product evaluation is concerning: since batches differ by strength and size, the model should be tested on a held-out product code to avoid overfitting to specific operating points.

- **Forecast & classification decoupling:** Forecast errors directly influence classification. The pipeline does not account for forecast uncertainty: all forecasts are taken at face value. Incorporating the LSTM's confidence (e.g. by predicting multiple scenarios or using a Bayesian approach) could prevent over-reliance on noisy forecasts.

- **No redundancy or safety net:** In practice, one would compare the model's alert to simpler rules (e.g. SPC limits, control charts) as a sanity check. If the ML model fails (as XGBoost did), the system should fall back to conventional triggers.

- **Feature usage:** The features use only the last 30 minutes of input and the forecast. They omit any historical quality or lab data (which may correlate with defects). For instance, API moisture or prior batch yields might add signal but are ignored. Environmental factors (e.g. humidity, temperature) also may affect product quality but aren't mentioned. While real-time lab data may not be available, any upstream indicators should be considered if timely.

- **Process constraints:** The code does not enforce any safety constraints on the forecast (for example, if fill depth is predicted out-of-range, that itself is a red flag). Embedding pharma limits (e.g. compaction force bounds from pharmacopeia [28]) could immediately flag impossible forecasts.

- **Imbalanced cost:** The evaluation metrics are symmetric (AUC, F1) but do not reflect actual cost. A few additional false positives (slowing lines) may be worth many fewer false negatives. The reward function in the RL section (not active here) hints at this trade-off [29]. The classifier should be tuned with a similar cost-aware objective.

Recommendations for Improvement

- **Threshold tuning and calibration:** Use the **ROC and precision–recall curves** to select an optimal probability threshold that meets business requirements. For example, to achieve >90% recall, accept a higher false-positive rate; conversely, to limit stoppages, raise threshold but verify recall stays adequate. Tools like the Youden's J statistic or domain costs can guide this. Probability calibration (Platt scaling or isotonic regression) might also help align predicted probabilities with true defect rates.

- **Model choice and ensembling:** Given the poor XGBoost result, try alternative classifiers. The sklearn GradientBoosting did significantly better here; also consider LightGBM or CatBoost with careful tuning. A **cost-sensitive or hierarchical model** (first a broad defect/no-defect, then severity classifier) could be valuable. Neural network classifiers (e.g. feed-forward or LSTM classifiers) might exploit temporal patterns directly and could even integrate forecasting internally. Ensembles of diverse models (XGB + GBM + RandomForest) can be combined (majority vote or stacking) to improve robustness.

- **Feature engineering:** Augment features with additional process context. For example, include *derivatives* or *trend rates* of each sensor (beyond the 5-min trend already used), or use sliding-window statistics (e.g. max drop in force). If available, incorporate upstream lab measurements (API moisture, impurity) or operator logs. Conversely, **prune redundant features**: many "current" vs "forecast" stats are correlated. Simplifying the feature set (via PCA or recursive feature elimination) could yield a simpler model that generalizes better and is easier to validate.

- **Forecast modeling:** Experiment with advanced sequence models. Convolutional networks, Temporal Convolutional Networks (TCN), or Transformers may capture long-range dependencies more efficiently than stacked LSTMs. Alternatively, reduce forecasting horizon (30 min) if accuracy suffers –

perhaps two-step forecasting (predict 15 min then another 15) or shorter windows for more reliable short-term forecasts. Multi-output regression with uncertainty quantification (e.g. Monte Carlo dropout) could allow the classifier to know "how sure" the forecast is.

- **End-to-end learning:** Consider a single model that takes recent sensor history and directly outputs defect probability (e.g. an LSTM or CNN over the time window). This could replace the separate forecast+features step, possibly yielding better joint optimization. It would also eliminate errors compounding from the two-step process.

- **Robust evaluation:** Perform **time-series cross-validation** (e.g. leave-one-batch-out or rolling-origin). Test the model on entirely new batches or time periods to check stability. If certain products or seasons behave differently, consider stratified models or include one-hot product indicators. Regular re-training (model maintenance) will be needed as processes drift.

- **Domain and compliance considerations:** Ensure the model and data pipeline meet pharmaceutical validation standards (21 CFR Part 11 compliance for audit trails, validation of software). All features and decisions should be documented and version-controlled. The model should be explainable: e.g. maintaining feature importance analysis [18] and providing human-readable rules for alerts will help with regulatory approval.

- **False-positive handling:** Given the cost of stoppages, the alert system could incorporate a confirmation step (e.g. if one consecutive window triggers, then halt) or gradually escalate (yellow→orange→red). Integration with statistical process control (SPC) charts or operator alerts can provide secondary checks.

- **Integration:** In deployment, the LSTM forecasts and classifier must run in real time (1 Hz data stream). Model inference should be lightweight or deployed on edge/cloud with guaranteed latency. The suggested microservice architecture (Azure Data Factory/Event Hubs) [30] is sound, but also ensure fail-safe modes: if the model is unavailable, default to existing SPC rules.

- **Continuous monitoring:** After deployment, monitor model performance (alarms vs actual defects) and retrain periodically. Use additional metrics like **False Alarm Rate (FAR)** and **Positive Predictive Value (PPV)** in dashboards, as well as traditional metrics. The code's "Yellow Flag Analysis" stats [31] hint at operational outcomes; extend this to trending KPIs (detected defects per month, false stops).

Conclusion

The reviewed pipeline has the right **conceptual structure** for early defect detection in tablet manufacturing, but its implementation needs refinement. The LSTM forecasting appears partially effective but uncertain for some sensors, and the classification stage fails under the chosen threshold. Pharma practice demands near-zero misses for defects and rigorous validation. In summary, we recommend:
- **Revisit model thresholds and class weighting** to ensure high recall of defects (potentially accepting more false alarms).
- **Tune or replace the XGBoost classifier**, possibly using ensemble or neural models and optimizing hyperparameters for defect detection.
- **Expand and curate features**, including more context and pruning redundancies.

- **Consider direct sequence classification** as an alternative to the two-stage approach.
- **Enforce compliance best practices** in documentation, explainability, and model monitoring.

By addressing these gaps and incorporating process knowledge, the system can more reliably translate sensor data into timely quality alerts, improving batch yield and regulatory compliance.

Sources: Pipeline and data details are drawn from the provided project and dataset documentation [2] [32], and from the analysis of the actual code outputs (forecasting errors and classification metrics) [12] [26]. These reveal the current model performance and informed the above recommendations.

---

[1] [3] [13] [27] [28] [29] [30] [32] Project Documentation.pdf
file://file-NEuKMJVGWENpJ3jqrHytUD

[2] [4] [6] [7] Dataset Documentation.pdf
file://file-45rEFEjtF9KanPsi3cGmGc

[5] [8] [9] [10] [11] [12] [14] [15] [16] [17] [18] [19] [20] [21] [22] [23] [24] [25] [26] [31] code-pt-1.pdf
file://file-4yETJY6rhFpbA2LvfkxRyY