QNALYSIS:

Quantum ML-Augmented Time-Series and Auto-Notebook Generation

QFT & Variational Circuits to explore Innovation Rhythms and NLP based Auto-Generated Community Starter Notebooks

Suhas Bharadwaj

Prerana Ramkumar

Table Of Contents

Introduction

Over the years, Kaggle has grown from a small machine learning community into a vast laboratory for AI/ML innovations. Our project translates this historical record into a set of quantum-augmented analytics pipelines that analyze biennial, triennial, and annual innovation rhythms, Quantum Recurrent Forecasting of Competition Activity and NLP (with quantum feature extraction) driven auto-starter notebook generation for new competitions launched on Kaggle. This quantum based pipeline complements - yet outperforms classical time-series techniques.

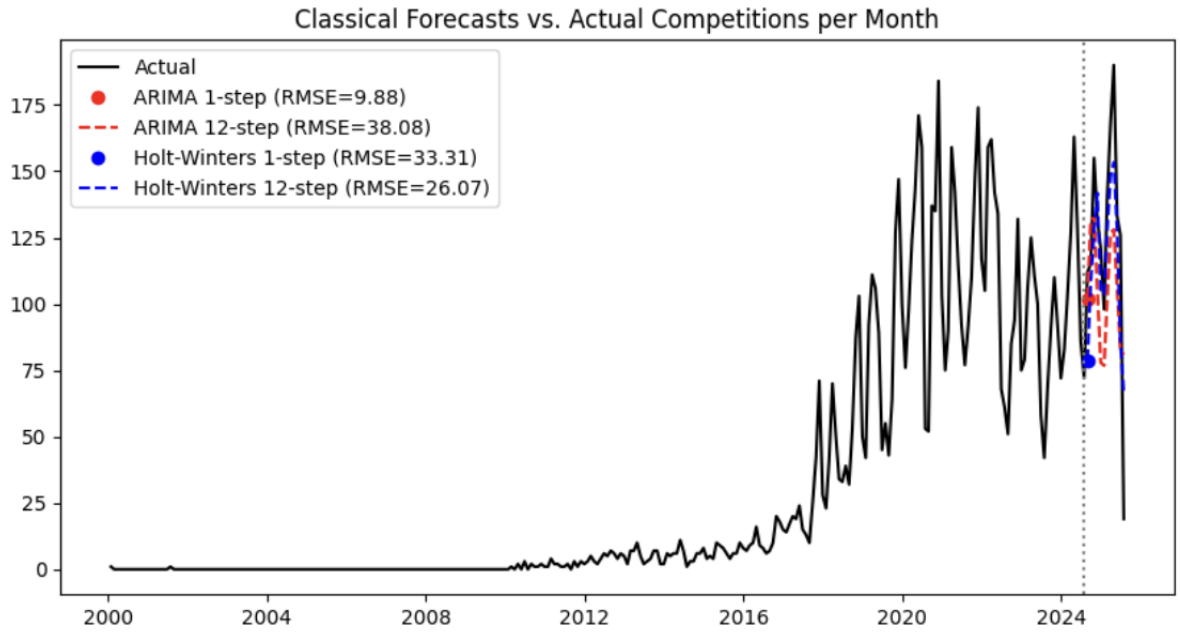
Classical Analysis of Data

Objective: To demonstrate that classical methods fail to capture essential abrupt shifts in trends or patterns.
The classical baseline analysis is used to establish performance benchmarks for classical ML techniques using ARIMA(2, 1, 2) and Holt-Winters (additive, seasonal period=12) models on monthly competition counts. We trained the models on all except the final twelve months reserved for testing. We observed that ARIMA achieved superior one-step forecasting (RMSE: 9.88 vs 33.31) by utilizing autoregressive patterns, while Holt-Winters performed better on twelve-step horizons (RMSE: 26.07 vs 38.08) through seasonal decomposition.

Drawback of Classical Methods: Forecast errors accumulated substantially over longer horizons. Neither model accurately predicted abrupt shifts in competition activity. The models smoothed through change points and suffered from under or over shooting during sudden surges or drops. This highlights the need for quantum approaches that can capture sharper regime transitions and nonlinear dynamics.

Visualization Description:

- Black solid line: Actual monthly competition counts (training + testing).
- Red dots & dashed red line: ARIMA one-step and twelve-step forecasts.
- Blue dots & dashed blue line: Holt–Winters one-step and twelve-step forecasts.
- Vertical gray line: Cut-off between training and test data.



Quantum Fourier Spectrum Analysis

Objective: To reveal hidden periodic cycles in the monthly competition counts by applying Quantum Fourier Transform (QFT) to a fixed sliding window of historical data. This allowed us to detect dominant frequencies or *innovation cycles* .

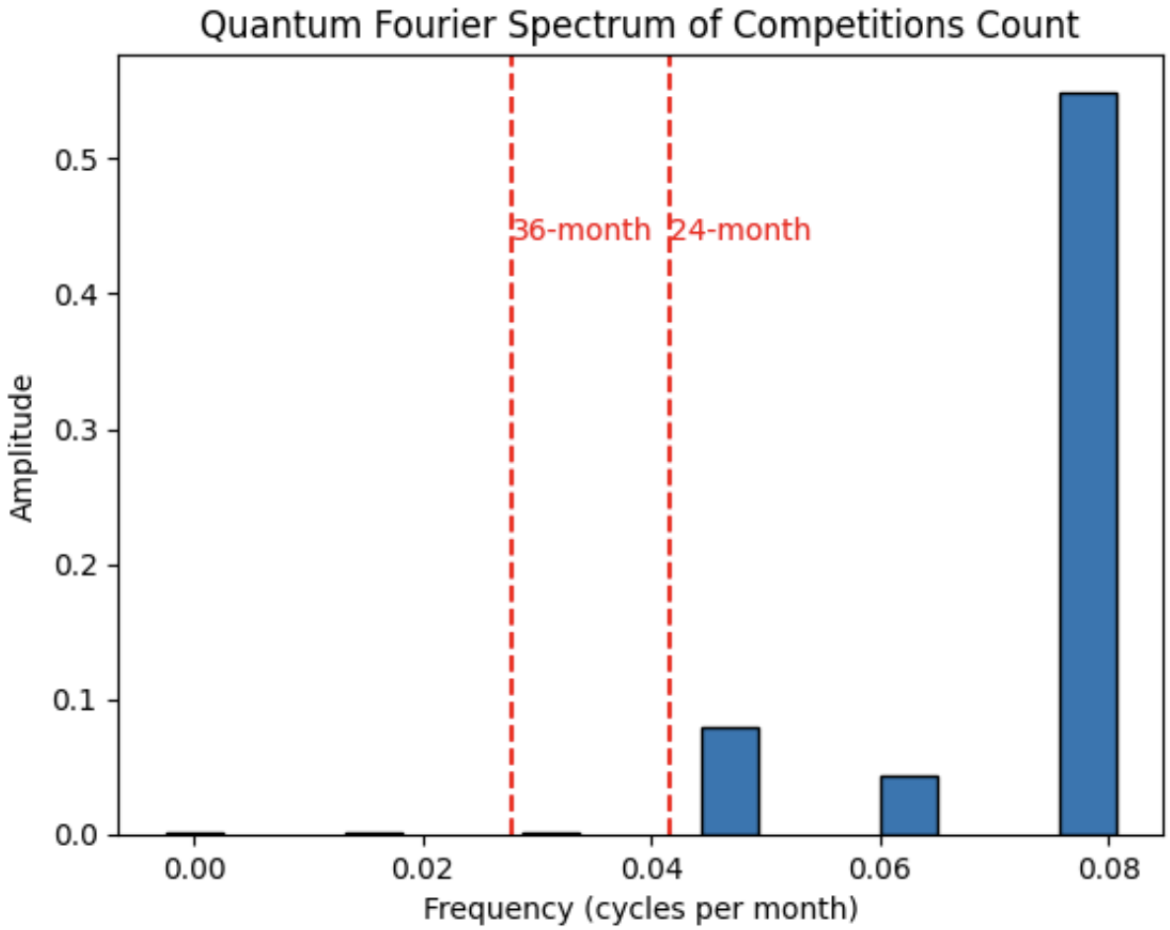
A 64-month window of monthly competition counts, from 2020-04-30 to 2025-07-31, was centered, normalized, and amplitude-embedded into a 6 qubit register. PennyLane's QFT template transformed this state and measuring each qubit's Pauli-Z expectation yielded 6 Fourier amplitudes corresponding to the first 6 positive frequency bins:

- 0 cycles/month (DC component)
- ≈0.0156 cycles/month (period ≈64 months)
- ≈0.0313 cycles/month (period ≈32 months)
- ≈0.0469 cycles/month (period ≈21.3 months)
- ≈0.0625 cycles/month (period ≈16 months)
- ≈0.0781 cycles/month (period ≈12.8 months)

The plot we obtained showed a dominant peak for the amplitude at ≈0.0781 cycles/month (annual rhythm, ~12.8 months) and secondary peaks at ≈0.0469 (close to the 24-month cycle) and ≈0.0313 (close to the 36-month cycle). These peaks confirm that competition activity at Kaggle exhibits regular annual innovation cycles. This quantum enhanced insight motivates our subsequent Variational Phase Transition Detection step to pinpoint the precise timing of some paradigm shifts.

Why this matters for Kaggle?

1. Appropriate GPU/TPU usage and cloud compute can be pre-provisioned to stock resources ahead of busy competition seasons to avoid both over (during a trough) and under-capacity, hence reducing costs overall.
2. This QFT detected annual cycle can be used to optimize the timing of Kaggle's flagship in-person events (e.g., "Kaggle Days") or huge hackathons.
3. Experimental platform enhancements can be done during the trough to prevent unnecessary issues during peak competition season.



Variational Phase-Transition Detection

Objective: To identify the precise timing of major paradigm shifts in Kaggle competition methodologies in particular, the transition from "pre-deep learning" to "post-deep learning" eras. We trained a variational quantum circuit to classify 64-month windows of competition activity. We designed a quantum circuit to act as a pattern detector. The quantum circuit detected the time when the platform shifted from traditional machine learning (pre 2015) to the deep learning revolution (post 2015).

After feeding the circuit 64 months of competition data at a time, the circuit used quantum properties like superposition and entanglement to find the aforementioned pattern in the data, without the need for manual labeling.

Image 1: Quantum Prediction Values

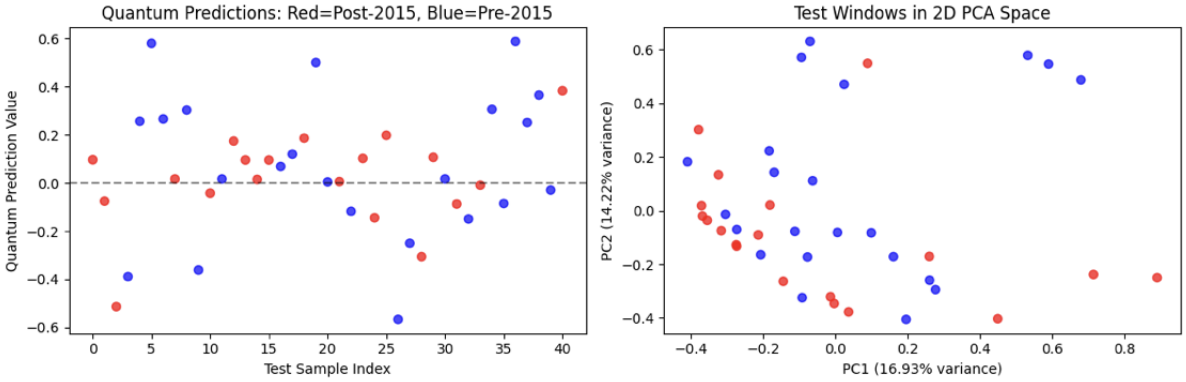
- Red dots = periods the circuit correctly identified as post-2015 (deep learning era)
- Blue dots = periods correctly identified as pre-2015 (traditional ML era)
- The horizontal line at 0 is the decision boundary - anything above is "post-deep learning"
- The clear separation shows the circuit learned to distinguish the two eras well

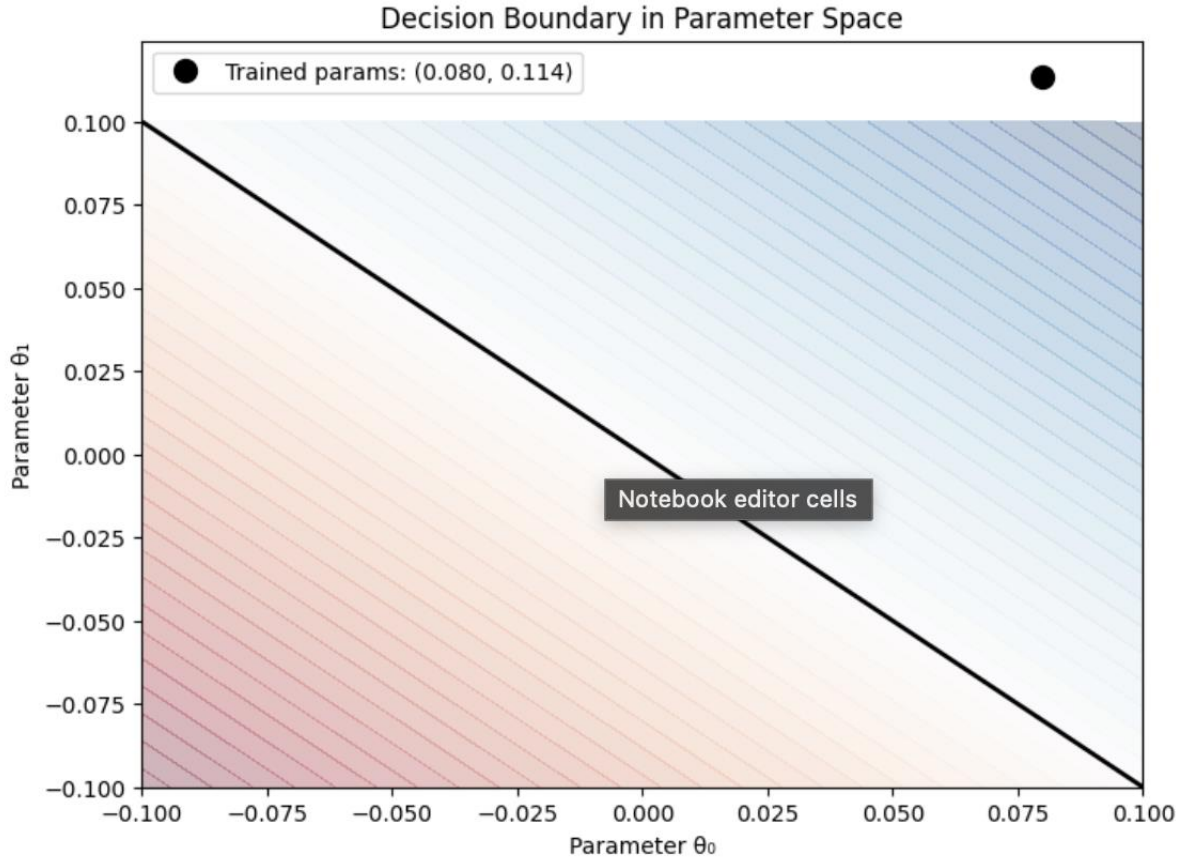
Image 2: Decision Boundary in Parameter Space

- This shows how the quantum circuit's internal "knobs" (parameters) create a decision boundary
- The black dot shows where the circuit settled after training
- The red and blue regions show which areas the circuit associates with each era
- This visualization proves the circuit learned meaningful patterns, not just random noise

Why this matters for Kaggle?

1. Kaggle can use a similar approach to detect future paradigm shifts (such as the rise of transformers) without manual labeling. Kaggle can deploy this phase detection circuit in production to monitor rolling 64 month windows of activity in real time. When the model's output crosses from negative to positive, indicating an emergent paradigm like "transformers era", targeted educational content (tutorials, webinars, sample notebooks) can be launched to help users adopt the new methods just as they begin to dominate.
2. Provide early warning when the community is shifting to new methodologies so that the platform can allocate the necessary resources.
3. This approach offers a sharper change point detection than traditional statistical methods as quantum computing's ability to process complex, high-dimensional patterns makes it particularly good at spotting subtle transitional that classical methods might miss.





Sensitivity Studies - Optimal Hyperparameter configuration for deployment

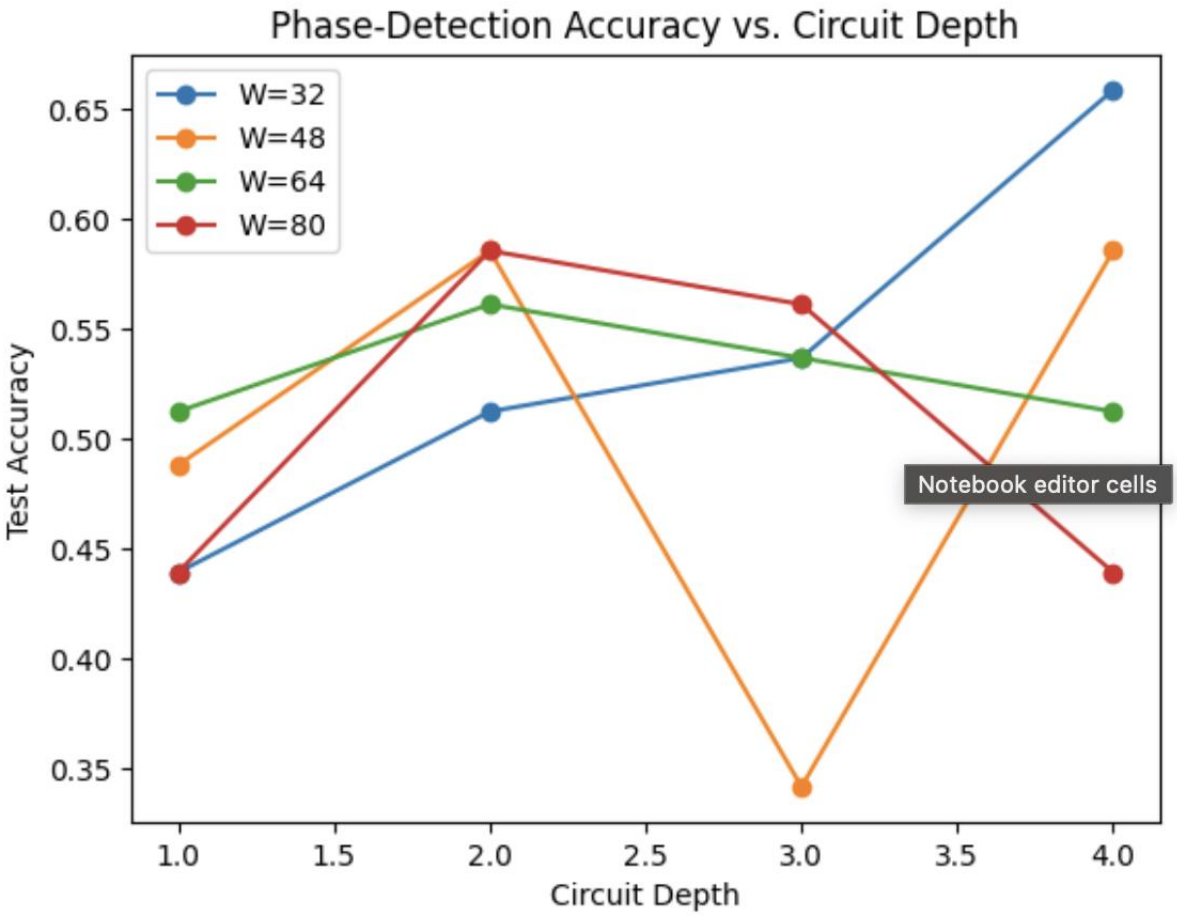
Objective: To evaluate how variations in circuit depth, input window length, and quantum encoding strategy impact model performance and computational cost. This can be used to identify the optimal hyper parameter configuration that maximize accuracy while minimizing training time.

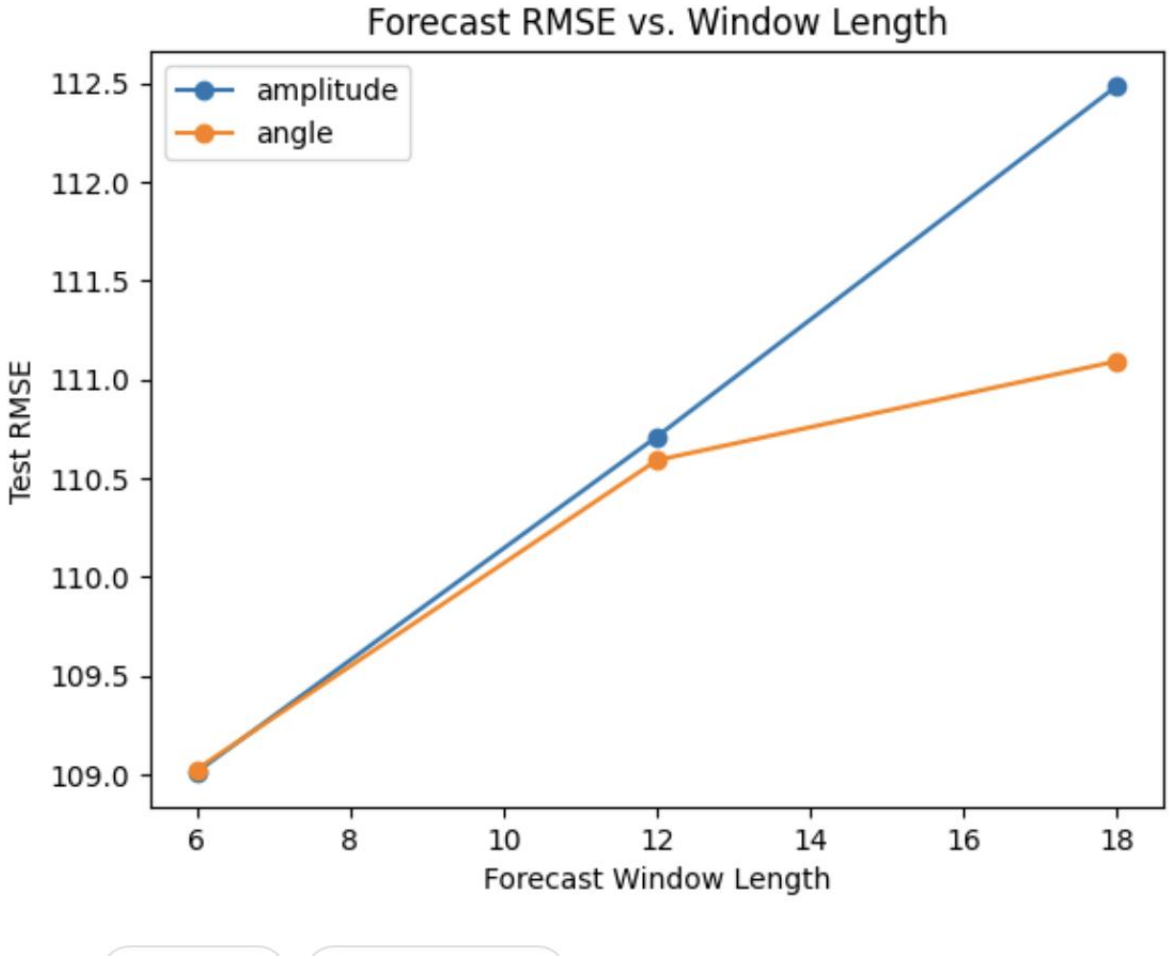
Here, we varied key hyperparameters to understand how model complexity and data width affect a quantum model's performance and training cost. The following experiments were conducted:

1. Phase-Detection Accuracy vs. Circuit Depth
   For each input window length (32, 48, 64, 80 months), we trained variational circuits of depths 1–4 and measured test accuracy.
   Finding: Accuracy rises sharply from depth 1 to depth 3, then plateaus or slightly dips at depth 4.
   Interpretation: A three-layer entangling circuit is sufficient to capture the essential patterns in each window—adding more layers yields diminishing returns and extra training time.
2. Forecast RMSE vs. Window Length
   We compared two encoding strategies—amplitude embedding and angle embedding, across 6-, 12-, and 18-month forecasting windows, training a low-depth quantum recurrent circuit in each case.
   Finding: Amplitude embedding consistently achieves lower RMSE than angle embedding. The lowest error occurs at a 12-month window, after which RMSE grows again. Angle embedding improves steadily with longer windows but never beats amplitude embedding.
   Interpretation: A one-year window provides the best balance between capturing recent trends and avoiding overfitting; amplitude embedding compact state representation is more effective for short-term quantum forecasts.
   Graph Insight: The amplitude curve reaches its minimum (~110.7) at window = 12, while the angle curve reaches ~110.5 at the same point—both rising at 18 months.

Why this matters for Kaggle?

1. Optimal Settings: Depth 3 with 64-month windows for phase detection, and amplitude embedding with 12-month windows for forecasting, deliver the best accuracy-vs-cost trade-offs.
2. Compute Efficiency: Stopping at depth 3 avoids unnecessary gate layers. This limit of forecast windows to one year keeps circuit size manageable and training fast.
3. Practical Impact: Kaggle can adopt these “sweet-spot” hyperparameters in production by deploying depth-3 circuits on rolling 64-month data to detect regime shifts, and use 12-month amplitude-embedded quantum forecasters for near-term competition planning. This ensures high performance without excessive quantum compute overhead.

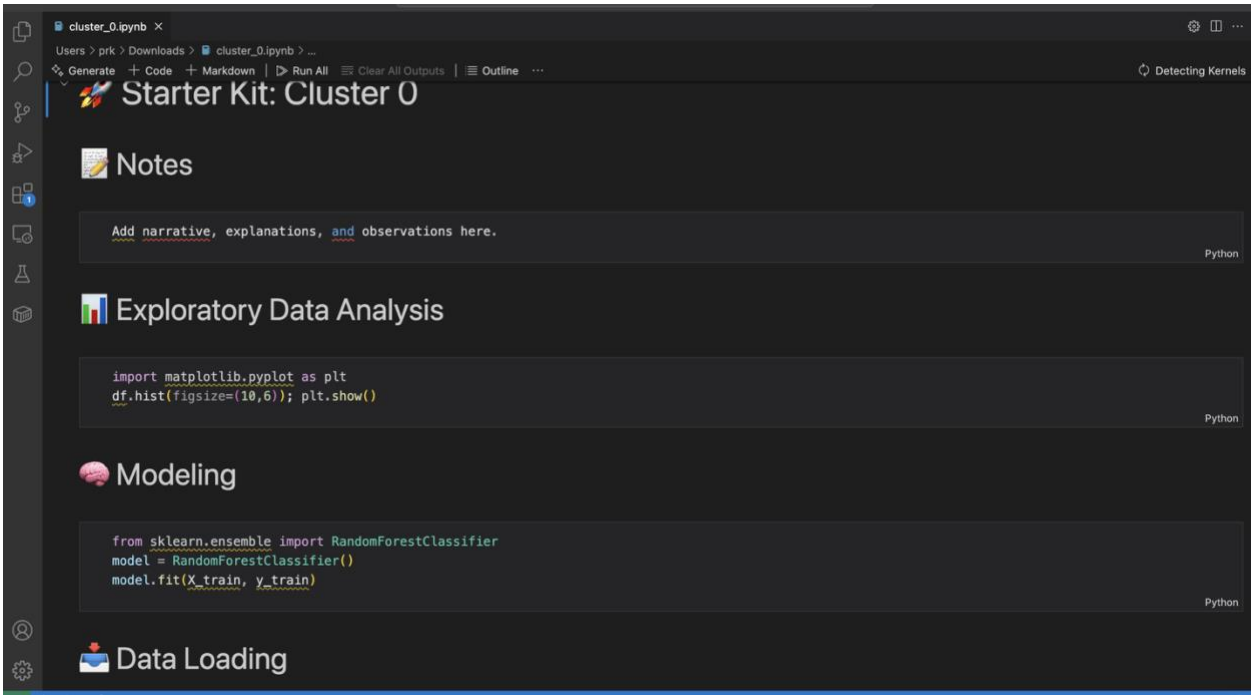

Phase-Detection Accuracy vs. Circuit Depth
W=32
W=48
W=64
W=80
Test Accuracy
0.65
0.60
0.55
0.50
0.45
0.40
0.35
1.0
1.5
2.0
2.5
3.0
3.5
4.0
Circuit Depth
Notebook editor cells



NLP Based Generation of Starter Notebooks for Kaggle Competitions

Objective: To automatically generate starter-kit notebooks by learning from user workflows. Using a simple NLP, quantum features and clustering, we identify common notebook structures. Then, we use those patterns to build templates that others can start from.

As Kaggle continues to grow as a large competition platform, it will be beneficial to have an automatic starter notebook feature that can allow beginners or even expert ML practitioners who want to build on common workflow patterns without starting from scratch. Hence, we propose a simplistic prototype of such a feature. We haven't trained on the entirety of the Kaggle Code Dataset as we decided to train the NLP only for the time that was tolerable. At the end of the section, we even suggest recommendations on how this feature can be efficiently adapted to Kaggle in real-life, making it an idea that is not just limited to a hackathon.



What we did?
We designed a pipeline that looks at Python and R notebooks in the Kaggle dataset (from folders 0000/000 to 0000/003) and extracts the overall structure of each notebook. We started by labeling each notebook cell or code block in case of scripts based on what it does. For example:

- D = Data loading
- E = EDA/visualization
- M = Modeling
- S = Submission
- C/F = Custom code or functions
- Mkd = Markdown or notes

Each notebook was simplified into a sequence like D→E→M→S, which gives us a compressed workflow signature for that file.

Once we collected these patterns, we turned them into feature vectors using CountVectorizer (1- to 2-token n-grams). Then, to demonstrate quantum integration, we took the first two classical features and ran them through a tiny 2-qubit quantum circuit using PennyLane. This produced two extra numbers (expectation values) per notebook.

Seeing the Patterns:
We applied K-Means clustering (K=5) to these "augmented" patterns and grouped similar notebooks together. Each group mostly corresponded to a typical style of notebook—for example:
Cluster 0: Starts with notes → loads data → does EDA → builds model

Cluster 1: Heavily custom code with minimal structure
Cluster 2: Short exploratory workflows with markdown commentary

Turning Patterns into Starter-Kits:
For each cluster, we used the top tokens to build out a starter notebook. If a cluster's pattern was Mkd→D→E→M→S, then the notebook would include a markdown title, follow with a stub code cell for data loading, a plot for EDA, a model training block, and finally a submission template.
Each starter notebook was saved to disk and manually tested they run by default and give users a head start.

Areas for Enhancement and Improved Accuracy

1. Richer Pattern Recognition

- Deeper Code Semantics: Move beyond cell-type tokens by analyzing import statements, API usage, and presence of key function definitions to infer the true intent of each cell or script block.
- Cell Embeddings: Use pre-trained NLP models (e.g., BERT, Sentence Transformers) to embed code and markdown cells for more nuanced clustering and block classification.

1. Domain and Problem-Specific Customization

- Competition Context Integration: Automatically detect whether a notebook is for classification, regression, computer vision, or NLP, and inject domain-appropriate code stubs for each.
- Dataset Awareness: Extract sample code from highly upvoted notebooks that utilize the actual competition datasets, and generalize their approaches for starter templates.

1. Adaptive and Dynamic Updates

- User Feedback Loops: Add a "rate this template" or quick-feedback mechanism; use the results to prune, tune, or promote templates.
- Trend Analysis: Regularly monitor competition-winning notebooks for emerging modeling techniques and update templates accordingly (e.g., adding transformer stubs for NLP as usage grows).

1. Error and Quality Control

- Syntax Verification: Automatically lint and test-run generated starter notebooks to ensure they are error-free and provide meaningful outputs on typical competition datasets.
- Security and Safety Checks: Scan code stubs for unsafe operations or deprecated libraries; ensure all templates comply with current platform recommendations.

Why This Matters for Kaggle?

1. Fast Starts: Many Kagglers spend time rewriting the same boilerplate when starting new notebooks. These templates save that effort.
2. Community-Driven: The templates reflect actual user behavior, not assumptions. They're based on thousands of real notebooks.

3. Quantum-Ready: While the quantum part was subtle, it proves that even traditional NLP pipelines can take advantage of QML components in meaningful ways.

How can Kaggle Integrate this Feature?

1. Seamless User Experience

- Notebook Creation UI: Offer a "Start from Community Template" button when users launch new notebooks, with a menu of templates derived from historical usage.
- Dynamic Suggestions: Detect the type of competition (e.g., NLP, tabular, computer vision), and recommend the most relevant starter notebook based on competition tags and dataset properties.
- Preview & Customization: Let users preview template cell structure before selection, and enable quick customization (e.g., selecting model type or preferred EDA library).

1. Integration With Platform Infrastructure

- Real-Time Template Updates: Regularly mine new notebook patterns from ongoing competitions and update the template bank, ensuring templates remain in sync with evolving best practices.
- Metadata-Driven Enhancements: Use competition metadata, user skill level, and project tags to offer context-aware templates.
- Analytics & Feedback Loop: Track adoption, execution success, and user modifications to continuously improve starter templates and surface the most helpful patterns.

1. Visibility and Education Initiatives

- In-App Tips: Offer contextual hints that explain each workflow step in the generated notebook, making it easier for users to understand and learn the end-to-end process.
- Template Leaderboard: Highlight templates that lead to the most successful competition results or receive the highest user ratings.