Mentor Project Report

Project: Integrating Quantum SVD & LSTM for Improved Recommender Accuracy
Author: Artem Churilkin
Mentors: Asel Sagingalieva, Arseniy Senokosov

1 Introduction and Objectives

Matrix factorization methods (such as SVD, NMF, ALS) have long been employed in recommender systems, providing robust and consistent results. However, these classical methods have limitations regarding accuracy and computational resources, despite supporting parallel computations. This becomes especially significant with the increasingly large datasets used in modern services, where both the number of users and the amount of content grow exponentially. Quantum algorithms, notably Quantum SVD, theoretically enable faster computations (with polylogarithmic complexity in terms of data size). However, such approaches require rigorous empirical validation, as current simulator-based implementations are not yet ideal and may not always meet theoretical expectations. This naturally raises the question of whether quantum methods can realistically outperform well-established classical methods, and whether such an advantage could be further enhanced by employing hybrid schemes.

- **Specific Task:** Predicting user ratings in the MovieLens100k dataset, which consists of explicit user ratings for movies on a scale from 1 to 5.
- **Objective:** Provide a fair comparison between the hybrid Quantum+LSTM method and classical approaches.
- **Main goals of the project:**
  - Conduct a thorough and fair comparison of the effectiveness between classical methods (Truncated SVD, NMF, ALS) and variations of the Quantum SVD method.
  - Develop and validate a hybrid Quantum SVD + LSTM approach to correct errors from quantum predictions and potentially uncover hidden patterns overlooked by classical methods.
  - Determine whether employing QML for recommendation systems is currently justified, and explore the practical benefits of combining quantum methods with neural networks.

2 Project Summary

This project implements classical recommendation methods (Truncated SVD, ALS), a quantum-based method using Pennylane, and a hybrid approach that corrects quantum predictions using an LSTM network.

- **Data and baseline:** The MovieLens100k dataset was split into train/validation/test subsets (60/20/20%), ratings were scaled to $[0, 1]$, and interactions were represented using sparse CSR matrices. Classical methods included Truncated SVD (implemented via scikit-learn) and ALS (custom implementation based on the Yandex Handbook). Hyperparameter tuning was performed using Optuna.
- **Quantum SVD:** Due to the absence of QRAM, data embeddings were obtained through classical Truncated SVD ($U, V^\top$ matrices) and subsequently encoded using two distinct quantum circuits (approach №2 was selected for the hybrid method):
  1. SWAP-test (AmplitudeEmbedding + CSWAP) to estimate the dot product;
  2. Variational network using AngleEmbedding with layers of RY rotations and CNOT gates, measuring $\langle Z_0 \rangle$, followed by linear scaling and bias (s, b).

  Generating 20,000 predictions required approximately 270 seconds, significantly slower compared to classical methods. Moreover, the second approach produced predictions within a narrow range of [3.40, 3.64], close to the dataset's average rating.

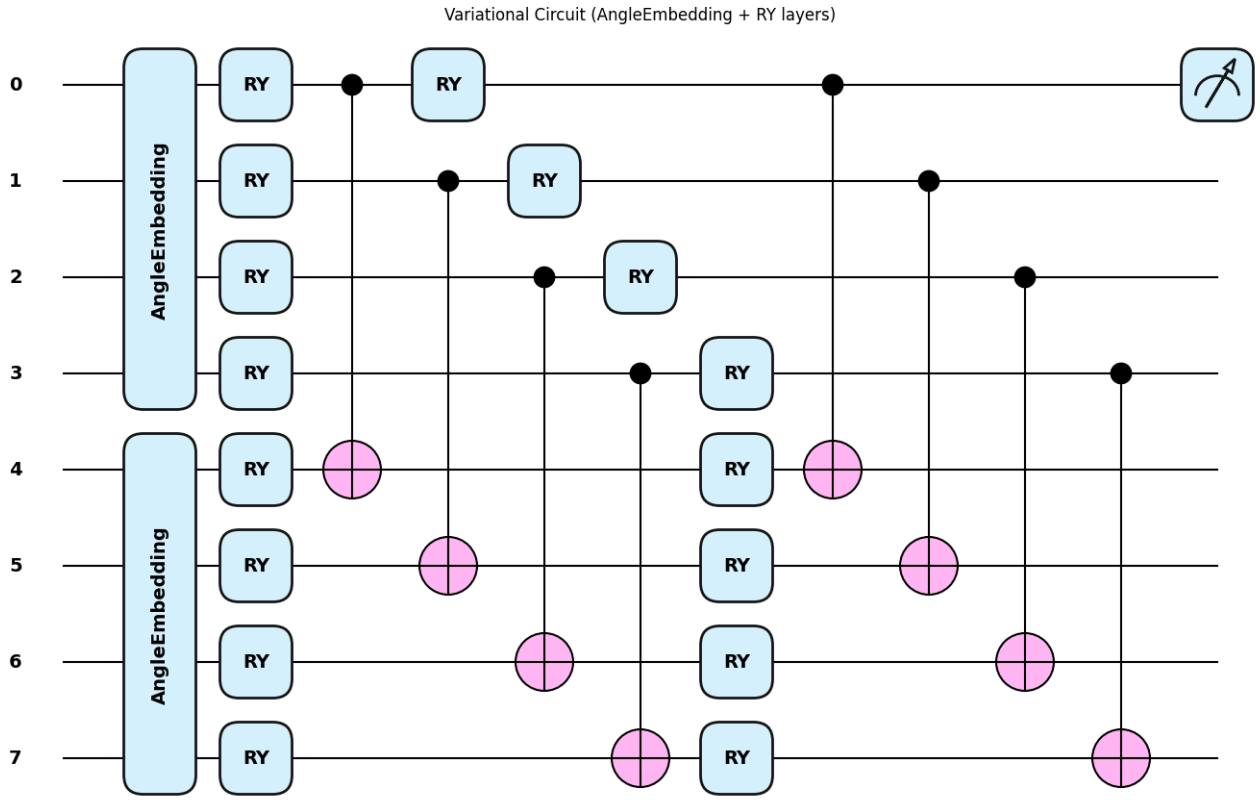

Рис. 1: Quantum circuit (approach №2) used in the hybrid method

- **Hybrid Quantum+LSTM:** The LSTM regressor received quantum predictions (r_q) from the AngleEmbedding network, as well as user and movie IDs, and was trained to predict the error $e = r_{\text{true}} - r_{\text{q}}$. The final rating

$$\hat{r} = r_{\text{q}} + \hat{e}$$

allowed for a wider prediction range (predicted error $\in [-3.2, 2.1]$), adding only a few hundred milliseconds of additional inference time. Despite this, RMSE improved slightly (from 1.12 in the purely quantum case to 1.09 in the hybrid approach).

3 Conclusions

- ALS demonstrated the best balance of quality and speed, achieving RMSE ≈ 0.70 and MAE ≈ 0.31 with an inference time of approximately 0.18 seconds. Therefore, based on these metrics, the chosen quantum methods were unable to outperform classical approaches.
- Quantum circuits (AmplitudeEmbedding, AngleEmbedding + variational network) proved functional but required hundreds of seconds to make 20,000 predictions. Additionally, they generated predictions within a very narrow range around the dataset's average rating, indicating suboptimal algorithm performance. Essentially, it appears that the quantum algorithms are simply aiming to minimize the metric naively, without effectively capturing individual user preferences, resulting in predictions close to the average rating.
- The hybrid AngleEmbedding + LSTM method successfully expanded the range of quantum predictions, slightly reducing RMSE (from approximately 1.12 to 1.09). This effectively improves recommendations from the standpoint of individual user preferences, as the predictions become more diverse and thus closer to reality, rather than merely approximating the average rating as the quantum-only schemes did. This highlights that raw numerical metrics alone may not fully reflect a model's practical effectiveness or success.

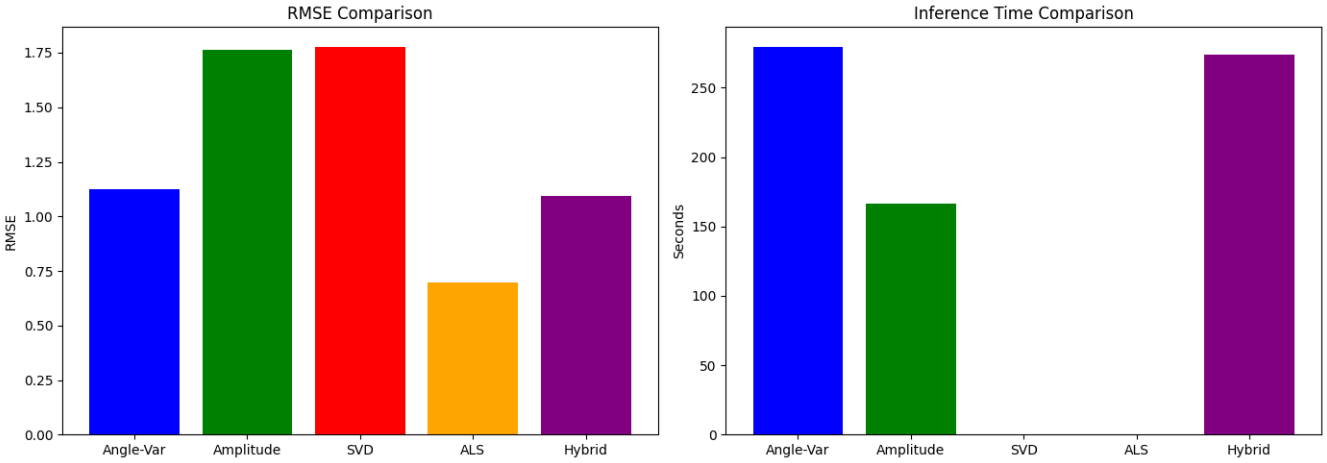

Рис. 2: Test RMSE and inference time comparison. ALS demonstrated the best overall performance.

- Training the LSTM solely on the quantum prediction error proved insufficiently informative. The model learned essentially a linear shift rather than directly predicting the ratings, thereby limiting the ability to discover nontrivial (hidden) user-item relationships.

4 Future Work and Potential Steps

- **Batching and Parallelization of QNodes:** Implement batch processing of queries in Pennylane and distribute simulations across multiple threads or computing nodes. This will reduce runtime and bring quantum methods closer to the speed of parallelized classical approaches.
- **Quantum Circuit Optimization:** Revise circuit topology (e.g., reduce circuit depth and the number of CNOT layers), and consider exploring quantum autoencoders.
- **Revising the LSTM Correction Task:** Train the neural network directly on predicting the final rating $\hat{r}$ rather than the error. Additionally, alternative architectures (e.g., GRU, Transformer) and loss functions (e.g., MAE, Huber) should be tested. Such an approach could potentially uncover more complex latent relationships and enhance prediction accuracy.
- **Scaling Up and Testing on Real QPU Hardware:** Evaluate the proposed methods on larger datasets (MovieLens1M, Netflix), and test the quantum circuit on cloud-based quantum processing units (QPUs) to measure real execution times and assess the impact of hardware noise.
- **Cold Start and Model Ensembles:** Utilize quantum-derived latent representations to address cold-start issues for new users or items, and create ensemble methods combining ALS, Quantum, and ML-based corrections to improve robustness and overall accuracy.

5 Papers that Inspired This Project

- The key paper by Kerenidis & Prakash introduced significant theoretical speedups for matrix processing, serving as the foundational work from which this project originated.
- Recent research on enhancing accuracy using an LSTM architecture (Yikai Mao, Shaswot Shresthamali, Masaaki Kondo) inspired the decision to incorporate LSTM networks on top of Quantum SVD predictions to correct errors.
- A comprehensive overview by Giovanni Semeraro & Marco Polignano describes major recommender methods as of 2022, providing valuable insights into current trends and methods in quantum-based recommender systems.
- P.S.: After starting work on this report, I encountered an intriguing paper from Skoltech and Sber AI Lab, which compares various recommender system methods. Figure 8 of this paper illustrates that the EASE algorithm surpasses ALS, suggesting that future quantum algorithms could beneficially be benchmarked against this method.

6 Appendix (Technical Details)

The main report concludes in Section 5. The following sections provide deeper insights for readers interested in a more detailed understanding of the matrix factorization methods and model architectures used in this project. A notebook with the full implementation is available at this link.

6.1 Dataset Processing and Baseline

For classical methods, data preprocessing was conducted by adopting sparse matrices, as the MovieLens100k dataset itself is highly sparse. Initially, row normalization using the L2 norm was applied; however, this caused an issue: models incorrectly interpreted zero values as low ratings, distorting the results (RMSE values became artificially low, not accurately reflecting true performance but rather an artifact of data preprocessing). Consequently, the L2 normalization was abandoned in favor of simply rescaling ratings from the original range of $[1, 5]$ to the interval $[0, 1]$. The dataset was split into three subsets: train (60%), validation (20%), and test (20%). Interaction matrices were represented in the CSR (Compressed Sparse Row) format, optimizing memory usage and computational efficiency. Experiments were conducted with Truncated SVD (scikit-learn), optimizing hyperparameters via Optuna. Hyperparameters were selected based on validation RMSE, and MAE and inference time were also recorded for comparative analysis.

The use of NMF (Non-negative Matrix Factorization) from scikit-learn encountered difficulties due to issues handling zero-valued elements. Specifically, the built-in implementation treated zeros as informative ratings, and attempts to mask these zeros—even in sparse matrices—were unsuccessful. This resulted in distorted outcomes, rendering hyperparameter tuning ineffective (the metric values remained nearly constant regardless of parameter adjustments). Ultimately, NMF was discarded in favor of ALS (Alternating Least Squares).

The main idea behind ALS is the alternating optimization of user and item latent factors by solving regularized least squares problems (L2 regression). I implemented ALS from scratch following the guidelines provided by the Yandex Handbook, as existing libraries like implicit support only implicit feedback, while our dataset contains explicit ratings. Subsequently, hyperparameters (number of latent factors, regularization strength, and iterations) were optimized using Optuna. Results on the validation set confirmed ALS as a strong and stable baseline for explicit recommendation tasks, albeit computationally more demanding (highly sensitive to hyperparameters such as the number of latent factors, defining the dimensionality of the latent space for users and items, as well as the number of iterations) compared to classical Truncated SVD.

6.1.1 Matrix Factorization: Truncated SVD and ALS

Why choose SVD and ALS?

- **Truncated SVD:**
  - SVD is at the core of the quantum algorithm described by Kerenidis & Prakash, making it logical to compare the classical approach directly with its quantum counterpart.

    Key advantages include:
  - A well-established and intuitive method: it is implemented directly in the `sklearn` library, allowing for straightforward usage and rapid extraction of latent representations without additional effort (which I utilized directly).
  - Efficient performance with CSR matrices and scalable across both users and items, ideally suited for scalable datasets like ours.
- **Alternating Least Squares (ALS):**
  - ALS is one of the most widely-used methods in recommender systems, primarily due to its explicit L2 regularization.
  - Unlike NMF, my custom ALS implementation ignores missing (zero-valued) entries in sparse matrices, genuinely improving validation metrics and avoiding biases toward zero ratings.
  - Another practical reason: I already had prior experience with ALS. Given the project's focus wasn't optimizing classical methods, choosing ALS allowed me to save time, reallocating it to the quantum and hybrid methods instead.

As mentioned earlier, ALS directly optimizes known ratings with L2 regularization. This approach yields more precise predictions but incurs higher computational costs due to the repeated solution of linear equation systems for each user and item.

Truncated SVD (sklearn) This method relies on the classical Singular Value Decomposition of the rating matrix $R \in \mathbb{R}^{m \times n}$:

$$R \approx U\,S\,V^\top, \quad U \in \mathbb{R}^{m \times k},\ S \in \mathbb{R}^{k \times k},\ V \in \mathbb{R}^{n \times k},$$

where $k \ll \min(m, n)$ (this condition appears frequently in matrix approximation literature, including the Kerenidis & Prakash paper). The notebook utilizes `sklearn.decomposition.TruncatedSVD`, which accepts two main parameters: the number of components `n_components`—the rank used to approximate the original matrix—and the number of iterations `n_iter`. Code snippets include:

- `svd = TruncatedSVD(n_components=k, n_iter=T)`
- `U = svd.fit_transform(R_train_scaled)`
- `S = svd.singular_values_`, `V = svd.components_`
- Reconstruction: $\hat{R} = U\,\mathrm{diag}(S)\,V$.

Truncated SVD performs effectively on sparse CSR matrices and swiftly extracts latent user and item representations. However, it lacks built-in regularization. Latent features, later employed in quantum circuits, were extracted as:

$$\Sigma = \sqrt{S}, \quad \text{user\_lat} = U\Sigma, \quad \text{item\_lat}^\top = \Sigma V^\top.$$

Alternating Least Squares (ALS) ALS solves the optimization problem of minimizing the regularized sum of squared errors for known (non-zero) elements:

$$\min_{X \in \mathbb{R}^{m \times k},\, Y \in \mathbb{R}^{n \times k}} \sum_{(u,i) \in \Omega} (r_{ui} - x_u^\top y_i)^2 + \lambda \Big(\sum_u C_u \|x_u\|^2 + \sum_i C_i \|y_i\|^2 \Big),$$

where Ω is the set of known ratings, and C_u, C_i represent the number of interactions for user u and item i, respectively.

The ALS algorithm proceeds in two alternating steps until convergence:

1. With Y fixed, solve the normal equation for each user u:

$$x_u = \left(Y_{\Omega_u}^\top Y_{\Omega_u} + \lambda C_u I\right)^{-1} (Y_{\Omega_u}^\top r_u),$$

   where Y_{Ω_u} is the submatrix consisting of rows y_i corresponding to items in Ω_u.

2. With X fixed, similarly update each item vector y_i:

$$y_i = \left(X_{\Omega_i}^\top X_{\Omega_i} + \lambda C_i I\right)^{-1} (X_{\Omega_i}^\top r_i).$$

In other words, ALS alternates between optimizing user and item matrices iteratively until convergence is achieved.

The implementation in code is provided by the function `als_train(R, n_factors, reg, n_iter)`:

- Initialize matrices $U, X \sim \mathcal{N}(0, 1/n\_factors)$ and $V, Y \sim \mathcal{N}(0, 1/n\_factors)$.
- Iteratively update all rows of U and V for n_iter iterations.
- Optimize hyperparameters ($n\_factors$, λ, n_iter) using Optuna, minimizing RMSE on the validation set.

The above summary closely follows descriptions from the Yandex Handbook. Additionally, I must highlight that ALS is highly sensitive to the hyperparameters $n\_factors$ (latent dimensionality) and n_iter (number of iterations), as increasing these values significantly raises computational costs. However, the trade-off between computational effort and performance improvement for different hyperparameter combinations was not thoroughly explored in this project.

6.2 Quantum SVD

6.2.1 General Scheme (Kerenidis–Prakash)

The quantum SVD algorithm relies on representing user–item vectors as amplitude states of a quantum system. Theoretically, this enables the computation of latent vector inner products in polylogarithmic time, provided data is loaded into QRAM. However, since such luxury is not available in simulators, approximations were employed instead.

6.2.2 Approximation without QRAM

- Due to the absence of QRAM, classical `TruncatedSVD` was utilized to obtain matrices $U \in \mathbb{R}^{m\times k}$ and $V^\top \in \mathbb{R}^{k\times n}$.
- Two embedding approaches were compared:
  - *AmplitudeEmbedding*: Entire vector loaded into amplitudes of n qubits.
  - *AngleEmbedding*: Each component encoded as a rotation angle $RY(\phi_i)$ on separate qubits.

6.2.3 Implementation in Pennylane

- **SWAP-test (`swap_test`)**:
  - Utilizes the `default.qubit` device with $1+2n_{\text{lat}}$ wires: one control qubit (wire 0) and $n_{\text{lat}} = \lceil \log_2 k \rceil$ wires each for vectors u and v.
  - Two sequential `AmplitudeEmbedding`s performed for vectors u and v on wires 1–n_{lat} and $(1 + n_{\text{lat}})$–$(1 + 2n_{\text{lat}})$ respectively.
  - Hadamard gates applied to the control qubit (wire 0) before and after embeddings. Between these gates, `CSWAP` operations are executed for each pair (a, b), with $a \in [1, n_{\text{lat}}]$, $b \in [1 + n_{\text{lat}}, 1 + 2n_{\text{lat}}]$.
  - Final Hadamard applied to control wire 0, then probability of measuring state $|0\rangle$ is obtained:
$$p_0 = \Pr(0), \quad \text{swap\_test}(u, v) = \big(p_0, 1 - p_0\big).$$
  - Scalar product computed as: $\text{dot\_q}(u, v) = (2p_0 - 1)\,\|u\|\|v\|$.
- **Variational Circuit with AngleEmbedding (`var_net`)**:
  - Device configured with $2k$ wires, where $k = \dim(u) = \dim(v)$.
  - `AngleEmbedding` applied separately to $u_{\text{angles}} = u \cdot \pi$ (range $[0, \pi]$) and v_{angles}, both using rotations `rotation="Y"`.
  - Circuit consists of L layers of parameterized rotations:
$$\text{for } \ell = 0, \ldots, L-1,\ \forall q \in [0, 2k-1]:\ RY\big(\theta_{\ell,q}\big)\,.$$
  - After each RY layer, entanglement is introduced through `layer_entangle()`, applying CNOT operations across pairs (a, b), with $a \in [0, k-1]$ and $b \in [k, 2k-1]$.
  - Measurement expectation $\langle Z_0 \rangle$ taken on wire 0. Network output computed as:
$$\hat{r} = s\ \langle Z_0 \rangle\ +\ b,$$
    with s and b as learnable parameters (`scale`, `bias`).
- **Training**:
  - Batched optimization using batch size of 16 with `qml.optimize.AdamOptimizer(stepsize=0.05)`.
  - Loss function: RMSE $\text{rmse\_on\_batch} = \frac{1}{N}\sum(\,\hat{r} - r_{\text{true}}\,)^2$.

6.2.4 Challenges and Limitations

- Lack of QRAM in simulators necessitated using classical SVD within embeddings.
- Each call to the QNode incurs significant overhead due to conversion between PennyLane and classical interfaces.
- No batch processing support for multiple queries: quantum functions must be called individually.

6.3 Hybrid Approach: Quantum Predictions + LSTM

6.3.1 Idea: Training LSTM to Correct Quantum Prediction Errors

The quantum predictor generates coarse recommendations with a relatively high RMSE. The goal is to train an LSTM to predict the correction term $e = r_{\text{true}} - r_{\text{q}}$, yielding the final prediction:

$$\hat{r} = r_{\text{q}} + \hat{e}$$

which should ideally achieve a lower RMSE.

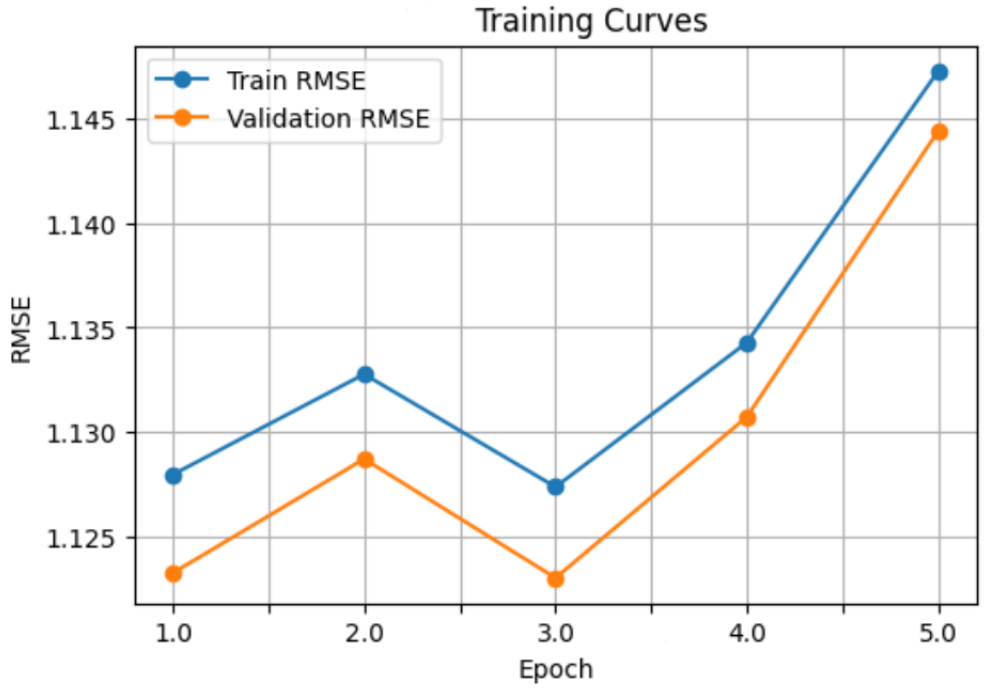

Рис. 3: Learning curves for `var_net` using AngleEmbedding. Note that the metric changes only slightly (in the second decimal place), with each epoch taking approximately 40 minutes. This behavior further supports the hypothesis that the algorithm primarily predicts an average rating rather than capturing individual user preferences.

6.3.2 LSTM Architecture (QuantumErrorLSTM)

- **Inputs**: Continuous feature `quantum_pred`, user ID, and item ID.
- **Embeddings**: User embedding $\text{Emb}_u(n_{\text{users}}, 32)$, item embedding $\text{Emb}_i(n_{\text{items}}, 32)$.
- **LSTM**: Input size $1 + 32 + 32 = 65$, hidden size 64, single-layer, `batch_first=True`.
- **Head**: Dropout(0.1) followed by Linear layer (64→1), predicting the error $\hat{e}$.
- **Loss function**: RMSE; optimization via Adam optimizer, learning rate 1e-4, early stopping based on validation RMSE.

6.3.3 Effect: Expanded Prediction Range and RMSE Reduction

- Quantum predictions initially had a narrow range of [3.40, 3.64] around the mean rating.
- LSTM-corrected ratings spanned a broader range, with predicted errors ranging from $[-3.19, 2.05]$ on the training set, moving away from average-based predictions. While not substantially improving RMSE numerically, this qualitatively enhanced user experience:

$$\text{RMSE}_{\text{Quantum}} \approx 1.12 \;\rightarrow\; \text{RMSE}_{\text{Hybrid}} \approx 1.09.$$

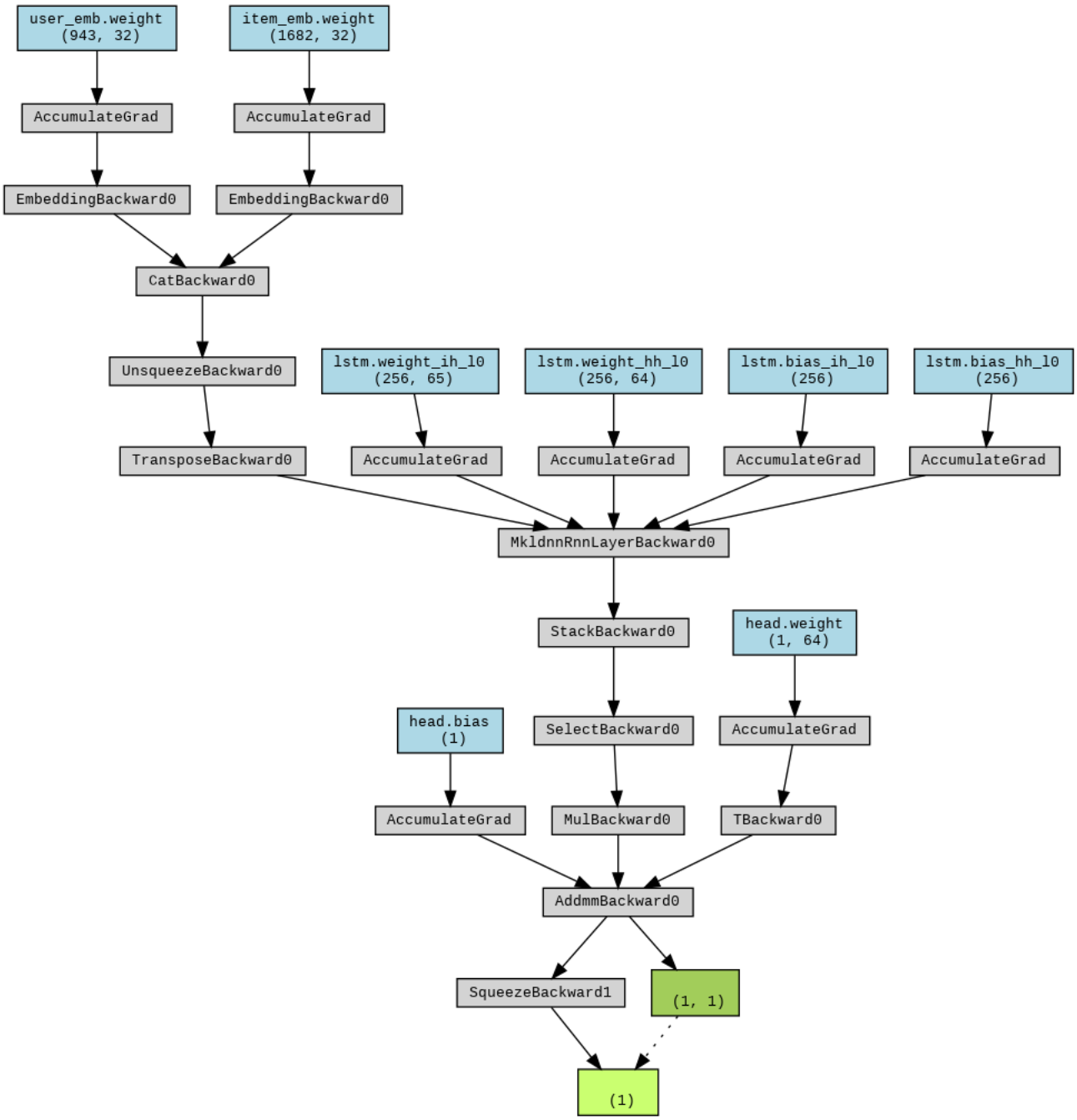

Рис. 4: Architecture of the employed LSTM network