Q-Tab: Quantized Tabular Data Generator

Julian Wustl*, Philipp Haid*, Yarema Okhrin, Claudius Schnörr

ICML International Conference On Machine Learning

TL; DR

Q-Tab is the first model to use **codebooks for expansion** beyond observed samples, not compression.

It learns a structured discrete numerical codespace via **Look–Up–Free Quantization** (LFQ) whose unassigned codes are trained to decode coherently through residual corruption. A **masked language model** (MLM) then learns realistic combinations of categorical, primary, and residual tokens for synthetic tabular data generation.

Why should you care ?

- **Immediate impact:** SOTA synthetic tabular data generation performance from structured discrete tokens.
- **Foundational model relevance:** A codebook-based tokenizer principle for tabular data.
- **Theoretical insights:** Corruption induces kernel regression; an online tracking view shows that corruption locality controls drift and variance.

Q-Tab Pipeline

Stage (1): Marginal standardization.

Numerical features are quantile transformed, whereas categorical features are one-hot encoded.

Stage (2.1): Joint numerical residual LFQ tokenization.

Q-Tab jointly encodes all numerical features as $h = E_\phi(x^{(\text{num})}) \in \mathbb{R}^{d_{\text{num}} \times B}, \quad \hat{h}_j = \alpha_1 b_j^{(1)} + \alpha_2 b_j^{(2)}, \quad \alpha_1 > \alpha_2 > 0.$

Each row $h_j \in \mathbb{R}^B$ is quantized into primary bits $(b_j^{(1)})$ and secondary residual bits $b_j^{(2)}$. During training, secondary bits are flipped independently with probability ε.

Stage (2.2): Learning token dynamics.

Each primary and each secondary bit sequence is assigned one token $(c_\cdot^{(1)}, c_\cdot^{(2)})$. The categorical realization are directly one hot tokenized, leading, for each observation, to the token sequence $[c_1^{(1)}, c_1^{(2)}, \ldots, c_{d_{\text{num}}}^{(1)}, c_{d_{\text{num}}}^{(2)}, z_1^{(\text{cat})}, \ldots, z_{d_{\text{cat}}}^{(\text{cat})}]$.

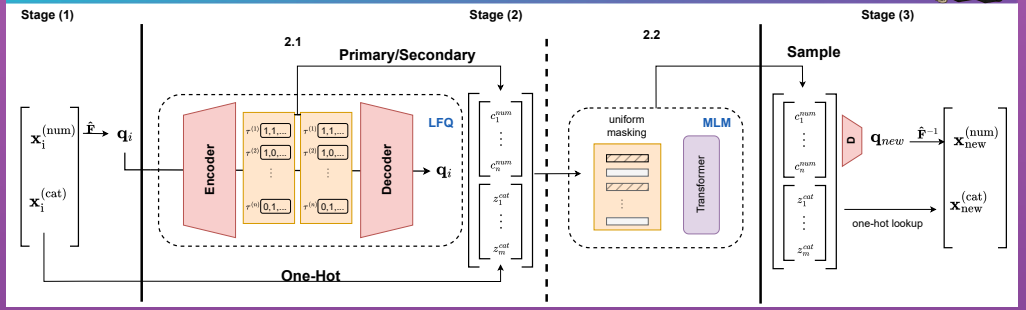
A BERT-style transformer learns the joint token distribution via an MLM objective. For secondary (residual) tokens, label smoothing spreads target mass from the observed token τ_{im} to the alternatives: $s_m(v) = (1-\delta)\mathbf{1}\{v = \tau_{im}\} + \delta \, \frac{\mathbf{1}\{v \neq \tau_{im}\}}{|\mathcal{V}| - 1}$.

This exposes the MLM to coherent alternatives beyond exact observed assignments, see 'LFQ as Online Tracking' on the right-hand side.

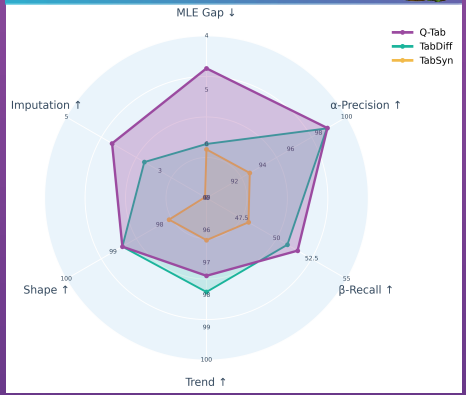
Stage (3): Synthetic data generation.

The sampling order of the tokens from the transformer matters: categorical tokens are sampled first, then primary numerical tokens and finally secondary (residual) numerical tokens.

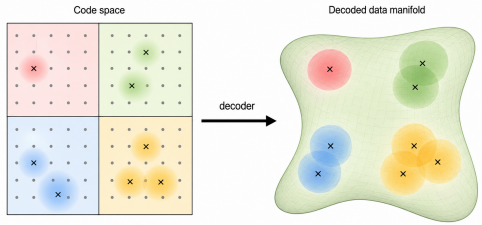
Main Design



Main Results

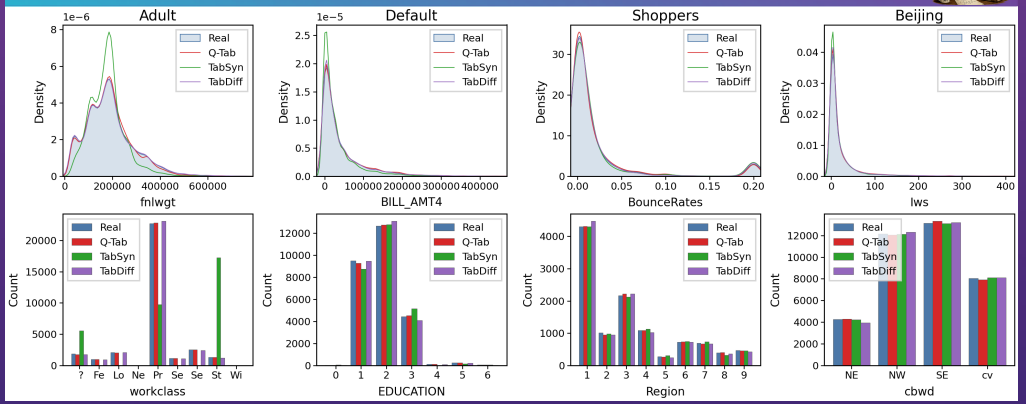


Codebooks for Expansion



- Assigned codes propagate supervision inside their orthant
- In overlap regions, non-assigned codes receive kernel-weighted barycentric targets
- Same-orthant neighborhoods overlap coherently
- Different outhants tend to remain separated

Single Column Densities



LFQ as Online Tracking

Problem.

Synthetic tabular generation requires a codebook with **more usable entries** than observed training rows (expansion). Otherwise, the generator can only replay a small set of decoded assignments which limits sample diversity. The core challenge is to make the extra capacity meaningful.

1. Corruption is kernel regression.

Let x_i be a training target, z_i its assigned code, and z a code reached through corruption $q(z \mid z_i)$. With fixed assignments, reconstruction under corruption yields $D^*(z) = \dfrac{\sum_i q(z \mid z_i) x_i}{\sum_i q(z \mid z_i)}$.

Thus corruption propagates assigned targets to non-assigned codes.

2. Joint training makes the target move.

Assignments $z_{i,t}$ move under joint encoder/decoder training, inducing a moving target regression problem.

For each code z in the fixed code space $\mathcal{Z}$, learning an unrestricted decoder constitutes a stochastic-approximation tracking problem: $X_t(z)$ samples a target whose current assignment can corrupt into z, with moving Bayes target $D_t^*(z)$. For the asymptotic tracking error at code z, which governs learnability (lower is better), we get

$$\limsup_{t \to \infty} \mathbb{E}\|e_t(z)\|_2^2 \lesssim \sigma^2(z) + \nu^2(z),$$

Where for each code z and iteration t, $\sigma_t^2(z) := \mathrm{Var}\big(X_t(z) \mid \mathcal{F}_t\big)$ is the **instantaneous variance** (how much the sampled target can scatter) and

$\nu_t^2(z) := \left\| D_{t+1}^*(z) - D_t^*(z) \right\|_2^2$ is the **target drift** (how fast the target moves in between iterations) with respective bounds in the equation.

3. Residual LFQ kernel geometry controls drift and variance.

Primary bits define orthants in shared latent space; Q-Tab fixes them and corrupts only residual bits. Hence supervision spreads locally within each orthant, but not across orthant boundaries.

$$\sigma^2(z) = \sigma^2_{\text{within}}(z) + \sigma^2_{\text{between}}(z), \qquad \nu^2(z) = \nu^2_{\text{within}}(z) + \nu^2_{\text{between}}(z).$$

For residual LFQ:

$$\sigma^2_{\text{between}}(z) = \nu^2_{\text{between}}(z) = 0.$$

Practical effect: The within-orthant terms become manageable after burn-in: encoder–decoder smoothness makes Q-Tab mix mostly compatible targets within each orthant, with weights determined by residual Hamming distance.