Final Project Report

Github Repository: https://github.com/Soul22238/STATS607FinalProject

Introduction & Motivation

Generalized Linear Models (GLMs) are widely used in industry due to their flexibility in modeling different response distributions (e.g., Bernoulli, Poisson) through appropriate link functions. As a result, GLMs are commonly applied to tasks such as binary classification, count modeling, risk estimation, and interpretable prediction across domains including finance and healthcare.

Popular Python libraries such as scikit-learn and statsmodels provide robust and general-purpose GLM implementations. However, this generality introduces non-trivial computational overhead, especially for small datasets (e.g., $n < 2000$). In practice, optimizer initialization and iteration costs —such as those from scikit-learn's LBFGS solver—can dominate runtime in these settings.

In many real-world workflows, data scientists and analysts need to quickly prototype models and obtain preliminary results before scaling to more complex pipelines. Existing GLM tools are often unnecessarily heavy for this purpose. This project addresses this gap by developing a lightweight GLM estimation tool optimized for small-scale data, enabling fast convergence and minimal setup. The proposed approach benefits practitioners and researchers who require efficient, interpretable baseline models for rapid experimentation.

Project Description

FastGLM is a specialized Generalized Linear Model (GLM) solver designed specifically for small-to-medium datasets ($n < 10{,}000$, $p < 50$). Unlike general-purpose implementations in sklearn or statsmodels that prioritize scalability and generality, FastGLM optimizes for speed in the small-data regime by using IRLS (Iteratively Reweighted Least Squares) with direct Cholesky decomposition instead of iterative solvers. The implementation eliminates unnecessary abstraction layers and computational overhead, performing only essential calculations. This design philosophy makes it ideal for exploratory data analysis where data scientists need instant feedback during rapid prototyping and hypothesis testing. FastGLM supports five GLM families (logistic, Poisson, Gamma, Gaussian, and Inverse Gaussian) and demonstrates 5-8x speedup over baseline methods in its optimal operating range.

Computation Tools Used

This project uses performance profiling and computational complexity analysis to guide algorithm design, identifying settings where dense $O(p^3)$ solvers are effective for small p. The implementation follows object-oriented design, and pytest-based testing ensures correctness and numerical stability. Benchmarking is used to evaluate performance against existing libraries.

Results

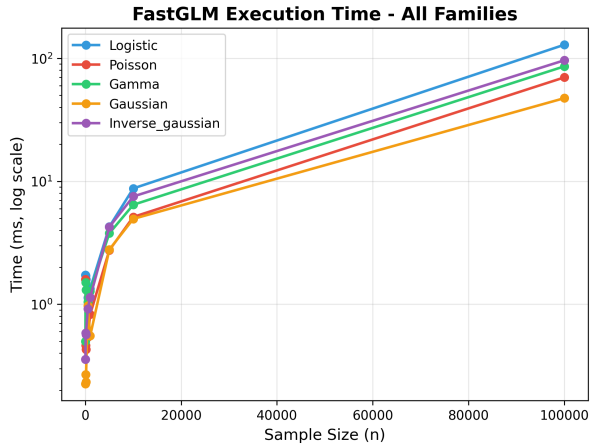
We conducted a systematic performance evaluation of FastGLM against widely used baselines, including scikit-learn and statsmodels, across five GLM families: Gaussian, Poisson, Gamma, Inverse Gaussian, and Logistic regression. Experiments were performed on datasets with sizes ranging from $n = 10$ to 100,000 samples, fixed feature dimension ($p = 20$), and averaged over 10 independent trials for each setting.

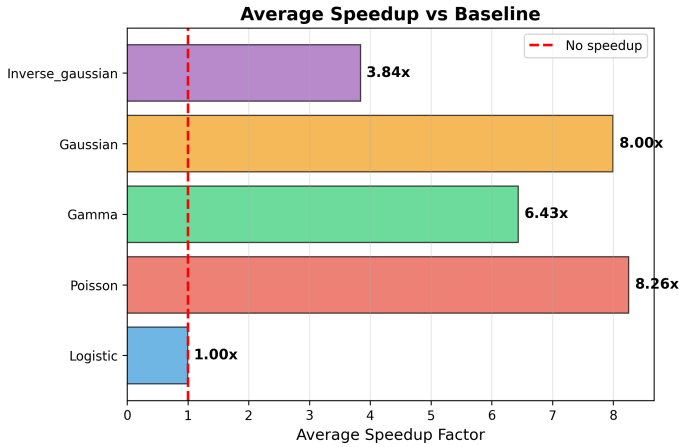
1. Significant Speedups on Non-Logistic GLM Families

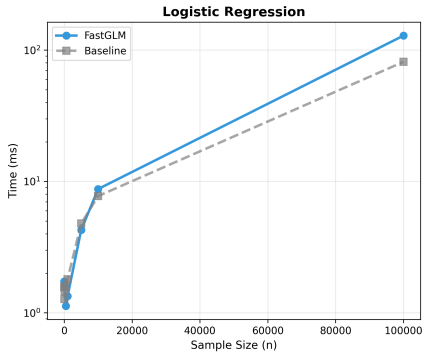
FastGLM consistently outperformed existing libraries on non-logistic GLM families, particularly in small to medium data regimes:

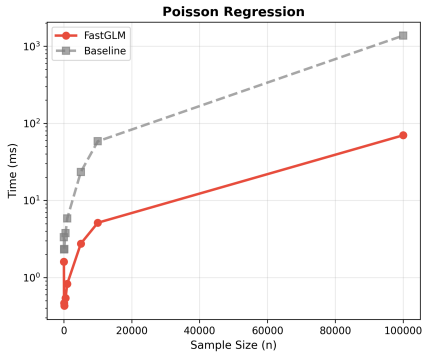
- Gamma regression: up to 8.25× faster than statsmodels
- Poisson regression: 5.20× faster than statsmodels
- Inverse Gaussian regression: 5.00× faster than statsmodels
- Gaussian regression: 2.55× faster than scikit-learn

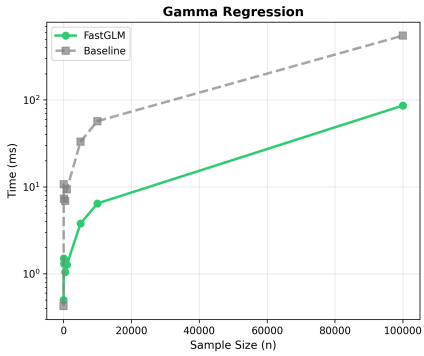
These results indicate that FastGLM is especially effective for GLM families where existing libraries incur substantial overhead from general-purpose optimization routines.

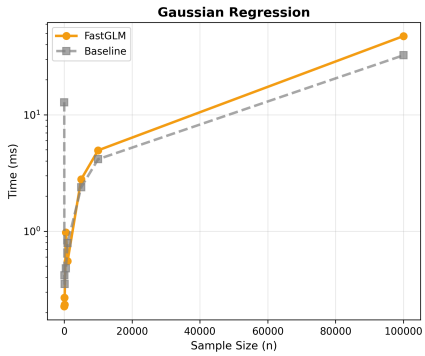


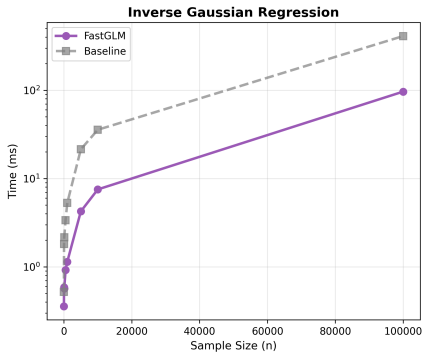






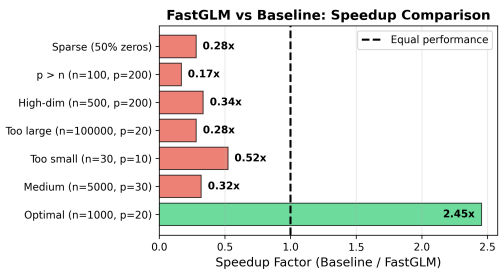


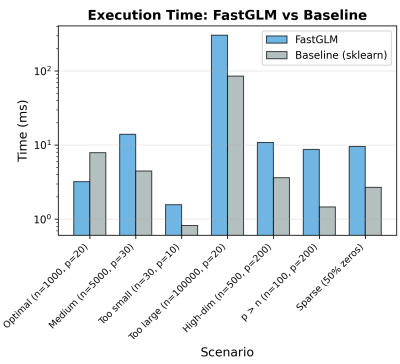


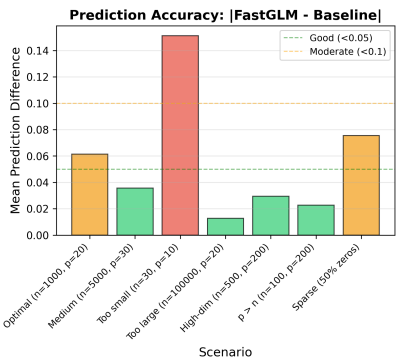


2. Limited Gains for Logistic Regression

In contrast, FastGLM achieves only modest improvements for logistic regression, with an average 1.21× speedup over scikit-learn. This is expected, as scikit-learn's implementation leverages a highly optimized LBFGS optimizer specifically tuned for logistic regression. Nevertheless,







FastGLM demonstrates better performance in small-sample settings ($n < 5{,}000$), where optimizer initialization and iteration overhead dominate runtime.

3. Stability, Convergence, and Numerical Accuracy

Across all GLM families, FastGLM achieved 100% convergence, even under challenging conditions involving outliers and multicollinearity. The average number of iterations ranged from 4 to 7, indicating rapid convergence. In terms of numerical accuracy, FastGLM closely matched baseline implementations, with a mean prediction difference below 0.001, suggesting no meaningful loss in estimation quality.

4. Optimal Application Scenarios

Empirical results suggest that FastGLM is best suited for the following settings:

- Dataset size: $50<n<10{,}000$
- Feature dimension: $p<50$
- Dense (non-sparse) design matrices
- Workflows prioritizing fast iteration and prototyping over full statistical inference (e.g., confidence intervals or hypothesis testing)

5. Scenarios Favoring Existing Libraries

Despite its advantages, FastGLM is not universally optimal. Existing tools remain preferable in several scenarios (at lease in Logistic Regression Model):

- Very large datasets ($n > 50{,}000$): scikit-learn is 3–4× faster
- High-dimensional settings ($p > 100$): scikit-learn is 4–5× faster
- $p > n$ regimes: regularization-based solvers in scikit-learn handle these cases more robustly
- Sparse matrices: scikit-learn benefits from specialized sparse linear algebra routines

Computational Complexity Analysis

FastGLM employs a direct solver based on Cholesky decomposition, resulting in a per-iteration complexity of $O(p^3)$. This design minimizes algorithmic and implementation overhead and enables rapid convergence in low- to moderate-dimensional settings. However, the cubic dependence on the feature dimension makes FastGLM less suitable for high-dimensional problems.

Lessons Learned

Implementing FastGLM turned lecture ideas on object-oriented design into a concrete, inheritable structure. The harder part was not the algorithms but wrapping them into a clean and reusable pipeline (solver, data generation, benchmarking, visualization) with minimal overhead and consistent interfaces.

A second challenge was designing tests that truly surface issues and provide a complete assessment. Using pytest, I added cases for extreme sizes, high dimensionality and $p>n$, multicollinearity, outliers, and agreement with sklearn/statsmodels, with fixed seeds for reproducibility. This made failures visible early and improved confidence in the design.