

Augmenting Stata with artificial intelligence

`xhdfe`: computational gains under econometric validation

Tiago Tavares and Miguel Portela

ISEG – University of Lisbon NIPE – University of Minho

2026 UK Stata Conference | London | 3 September 2026

Can AI improve on well established stata commands without changing the results?

Estimator call time for a 55.9-million-observation worker-firm-year fixed effects regression:

3 h 59 min

`reghdfe`
reference

3 min 22 s

`xhdfe`, CPU
71.2× faster

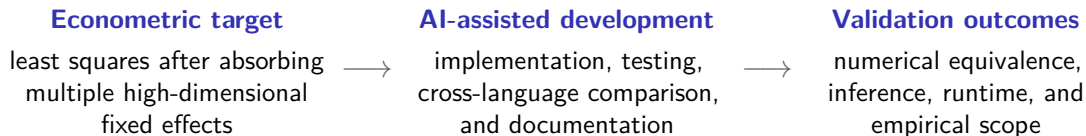
1 min 10 s

`xhdfe`, CUDA
206.3× faster

The speedup **changes computation...**but **not the econometric result**:

- use widely popular `reghdfe` as reference;
- both coefficients and clustered standard errors agree to numerical precision.

The case study of `xhdfc` links development inputs to observable outcomes



Implication: AI can be an input to the development process with corroborating evidence coming from tests and benchmarks.

Main question:

How can **AI-assisted software development** reduce the cost of implementing and optimizing an HDFE estimator **while preserving** its estimand, numerical criterion, inference, and **Stata interface**?

Roadmap:

- **development protocol**
- **estimator and architecture**
- **validation and performance**
- **companion commands and applications**
- **boundaries and conclusion**

Development protocol

How were AI-assisted changes proposed, tested, and accepted?

Econometric credibility extends from the estimand to its implementation

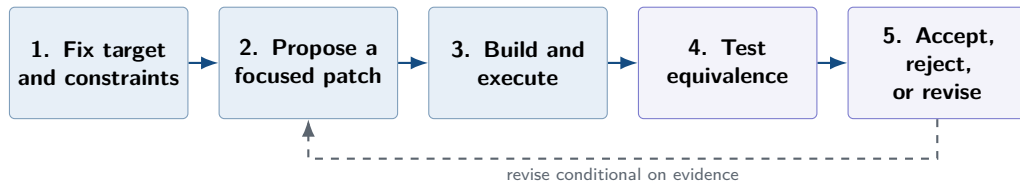
In the case of `xhdfe`, this involves:

- Translating the estimand, convergence rule, and inference into code
- Navigating a mixed Stata, C++, CUDA, Python, and R codebase
- Diagnosing numerical failures and edge cases
- Building tests and benchmark harnesses
- Keeping help files, examples, and releases synchronized
- Packaging across Linux, macOS, and Windows

Role of AI assistance

It **reduces the cost** of moving from an econometric specification to an **inspectable** implementation and a reproducible validation result.

Protocol for AI-assisted changes



Each proposed change was built, executed, and retained only after passing pre-specified checks.

AI lowers implementation and verification costs across software layers

In the case of `xhdfc`, this involves:

- **Codebase navigation:** connect ado parsing, plugin transfer, C++ kernels, and CUDA paths
- **Focused patches:** parsing, diagnostics, memory layout, numerical safeguards
- **Benchmark discipline:** launch runs, parse logs, compare evidence
- **External comparison:** inspect patterns in `reghdfc`, `fixest`, and Julia
- **Documentation:** synchronize help, examples, release notes, and paper text
- **Packaging:** trace build and runtime failures across platforms

Operationally, the contribution is cumulative: repeated, inspectable revisions across a version-controlled project.

Researcher choices define the estimand and acceptance region

Fixed by researchers

- estimator definition
- convergence targets
- acceptable numerical error
- release and performance claims
- economic interpretation

AI-assisted, conditional on tests

- implementation drafts
- local refactors
- test scaffolds
- benchmark parsing
- documentation drafts

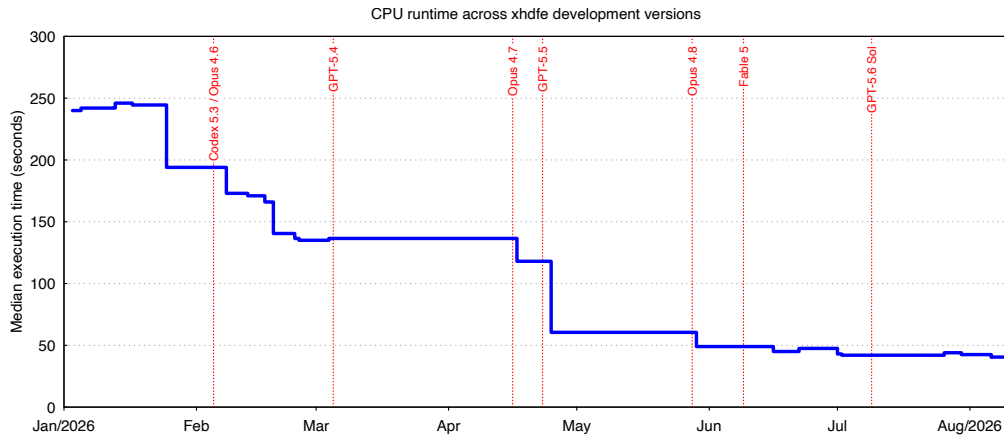
AI changes the cost of implementation. Researchers still define what constitutes valid evidence.

Acceptance protocol for candidate revisions

- 1 **Execute**: the relevant Stata, C++, Python, R, or CUDA path must run.
- 2 **Certify convergence**: satisfy the stopping rule and independent residual check.
- 3 **Match the reference**: sample, coefficients, VCE, degrees of freedom, and stored results must pass pre-defined tolerances.
- 4 **Benchmark**: hold data, hardware, settings, and backend telemetry fixed.
- 5 **Retain or reject**: a faster revision is discarded if any econometric constraint fails.
- 6 (Leave some discretion to the human designer)

Runtime enters the objective only after econometric equivalence constraints are met.

Measured CPU runtime over the development history of `xhdfc`



Restricted QP historical benchmark; 16 threads; ten repetitions where available.

The estimator target was fixed before optimization

- 1 Fix the reference estimator and stopping rule
- 2 Isolate the C++ absorption and regression kernel
- 3 Measure convergence, iterations, and memory
- 4 Expose the kernel through a Stata plugin
- 5 Restore syntax, inference, and postestimation
- 6 Accept optimizations only after validation

The statistical target and Stata behavior were fixed before runtime entered the objective.

Estimator and architecture

What does `xhdfe` compute, and how does its architecture preserve the requested model?

xhdfe changes the computation, not the regression model

At the Stata surface

- linear regression with multiple high-dimensional fixed effects
- familiar syntax and e-class results
- ordinary Stata postestimation

Under the hood

- compiled C++ absorption and regression
- threaded CPU execution
- optional CUDA for suitable workloads

The requested model remains familiar but the computational backend changes.

The standard installation is one command on every platform

macOS, Windows, and Linux

```
net install xhdfe, replace ///  
    from("https://raw.githubusercontent.com/reisportela/xhdfe-xfe/gh-pages/stata")
```

Optional CUDA build: Linux with an NVIDIA GPU

```
xhdfegpu  
discard
```

A familiar Stata specification requires only a command change

* Established workflow

```
reghdfe ln_wage age tenure educ, absorb(worker firm year) vce(cluster worker)
```

* xhdfe workflow

```
xhdfe ln_wage age tenure educ, absorb(worker firm year) vce(cluster worker)
```

- **Data:** the same Stata data remain in memory.
- **Specification:** absorbed dimensions and clustering are unchanged.
- **Postestimation:** results are returned through the usual e-class workflow.

The computational problem is projection, not dummy construction

We consider the typical linear high-dimensional fixed-effects model:

$$y_i = \mathbf{x}_i' \boldsymbol{\beta} + \sum_{g=1}^G \mathbf{d}_{ig}' \boldsymbol{\alpha}_g + u_i$$

with a least-squares target:

$$(\hat{\boldsymbol{\beta}}, \hat{\boldsymbol{\alpha}}) \in \arg \min_{\boldsymbol{\beta}, \boldsymbol{\alpha}} (\mathbf{y} - \mathbf{X}\boldsymbol{\beta} - \mathbf{D}\boldsymbol{\alpha})'(\mathbf{y} - \mathbf{X}\boldsymbol{\beta} - \mathbf{D}\boldsymbol{\alpha})$$

The objective is ordinary least squares. `xhdfe` applies the projection without constructing $\mathbf{D}$.

FWL leaves a small regression after the fixed effects are removed

Applying a Frisch-Waugh-Lovell (FWL) representation:

$$\mathbf{M}_D = \mathbf{I} - \mathbf{D}(\mathbf{D}'\mathbf{D})^{\dagger}\mathbf{D}', \quad \tilde{\mathbf{y}} = \mathbf{M}_D\mathbf{y}, \quad \tilde{\mathbf{X}} = \mathbf{M}_D\mathbf{X}$$

provides a low-dimensional regression solution:

$$\hat{\beta} = (\tilde{\mathbf{X}}'\tilde{\mathbf{X}})^{-1}\tilde{\mathbf{X}}'\tilde{\mathbf{y}}$$

Absorption

Apply $\mathbf{M}_D$ to the outcome and each regressor without constructing the dummy matrix.

Final regression

Estimate the retained slope coefficients using only the residualized variables.

Multiple fixed effects make absorption iterative

For categorical fixed effects, one applies group demeaning iteratively:

$$\mathbf{Z}^{(k+1)} = \mathbf{M}_G \cdots \mathbf{M}_2 \mathbf{M}_1 \mathbf{Z}^{(k)}, \quad \mathbf{Z}^{(0)} = \mathbf{Z} \equiv [\mathbf{y}, \mathbf{X}].$$

which converges to the desired projection:

$$\mathbf{Z}^{(k)} \longrightarrow \mathbf{Z}^* = \mathbf{M}_D \mathbf{Z}.$$

Numerical methods change the path and cost of reaching $\mathbf{Z}^*$, not the FWL target $\mathbf{M}_D \mathbf{Z}$.

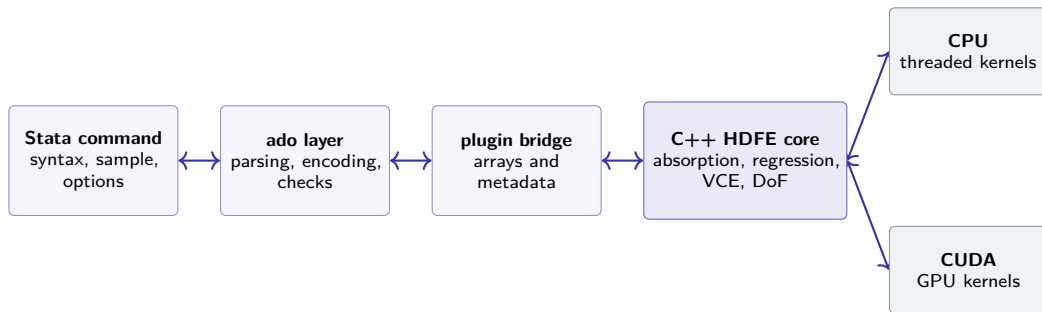
Why iterate?

Removing one set of group means generally reintroduces means in another fixed-effect dimension.

When to stop?

Stop when another full sweep changes $\mathbf{Z}$ by less than the pre-specified tolerance.

Stata stays at the surface while computation moves underneath



Compatibility is implemented at the Stata surface with performance engineering concentrated underneath.

Runtime falls through fewer sweeps and cheaper sweeps

1. Fewer sweeps to tolerance

- fixed-effect order selection
- extrapolated projection sweeps
- modified LSMR for eligible problems
- independent residual certification

2. Lower cost per sweep

- precomputed integer group indices
- joint in-place demeaning of y and X
- OpenMP CPU threading
- CUDA kernels for large workloads

Both routes preserve the stopping rule and the returned statistical results.

Stata, Python, and R call the same C++ estimator

Stata: familiar e-class workflow

```
xhdfe y x, absorb(firm year)
```

Python: formula or low-overhead array API

```
xhdfe.feols("y ~ x | firm + year", d)
```

R: fixest-style formula surface

```
xhdfe(y ~ x | firm + year, d)
```

Interfaces differ although the numerical core is the same.

Validation and performance

Does `xhdfc` reproduce the reference estimates, and how large are the runtime gains?

The benchmark covers 55.9 million observations in two regressions

- StataNow/MP 19.5; 16 threads; Intel Xeon Gold 6542Y; NVIDIA H100 NVL
- Portuguese matched employer–employee data (Quadros de Pessoal) from 1985–2020:
 - 55.9m observations, 5.95m workers, 799,265 firms, 36 years
- Common seniority (worker clusters) and firm-specific seniority (worker–firm clusters):

Common seniority:
$$y_{ijt} = \mathbf{p}(e_{it})' \beta + a_{ijt} \delta_1 + b_{ijt} \delta_2 + \underbrace{\theta_i + \lambda_t + \psi_j}_{\text{absorbed fixed effects}} + u_{ijt}$$

Firm-specific seniority:
$$y_{ijt} = \mathbf{p}(e_{it})' \beta + \underbrace{\theta_i + \lambda_t + \psi_j}_{\text{absorbed fixed effects}} + \underbrace{a_{ijt} \gamma_j + b_{ijt} \eta_j}_{\text{absorbed firm-specific slopes}} + u_{ijt}.$$

$\mathbf{p}(e_{it})$: quartic in experience; a_{ijt} : firm seniority; b_{ijt} : seniority spline.

CPU and CUDA reproduce `reghdfe` to numerical precision

Same sample, specification, clustering, and singleton rule in every comparison.

Specification	Backend	Max. $ \Delta\hat{\beta} $	Max. $ \Delta\widehat{SE} $
Common seniority	CPU	3.75×10^{-12}	1.45×10^{-13}
Common seniority	CUDA	5.32×10^{-11}	1.50×10^{-12}
Firm-specific seniority	CPU	3.08×10^{-10}	5.14×10^{-11}
Firm-specific seniority	CUDA	9.61×10^{-09}	1.26×10^{-10}

For common seniority, CUDA is $141\times$ faster

1 h 1 min 7.5 s

reghdfe
reference

1 min 29.6 s

xhdfe, CPU
 $41.0\times$ faster

25.9 s

xhdfe, CUDA
 $141.4\times$ faster

- The comparison uses the same sample, model, clustering, and singleton rule.
- Coefficients and clustered standard errors agree to numerical precision.

With firm-specific slopes, CUDA is $206\times$ faster

3 h 59 min

reghdfe
reference

3 min 22 s

xhdfe, CPU
 $71.2\times$ faster

1 min 10 s

xhdfe, CUDA
 $206.3\times$ faster

- Same sample, covariates, absorbed effects, two-way clustering, and singleton rule.
- Maximum coefficient difference: 3.08×10^{-10} on CPU and 9.61×10^{-9} on CUDA.

Users can trade a small amount of numerical precision for lower runtime

Both modes converged and passed the validation checks.

Mode	Backend	Seconds	Max. $ \Delta\hat{\beta} $
reghdfe-comparable	CPU	89.6	3.75×10^{-12}
reghdfe-comparable	CUDA	25.9	5.32×10^{-11}
xhdfe-fast	CPU	61.3	2.03×10^{-07}
xhdfe-fast	CUDA	22.7	2.03×10^{-07}

Reference-comparable mode

Prefer closer numerical agreement with reghdfe.

Opt-in fast mode

Prefer lower runtime under an explicit tolerance trade-off.

Comparison with other implementations (common-seniority model)

Implementation	Backend	Seconds	Speedup
<code>reghdfe</code> , Stata	CPU	3,667.5	1.0×
<code>FixedEffectModels.jl</code>	CPU	916.3	4.0×
<code>fixest</code>	CPU	486.5	7.5×
<code>pyfixest</code>	CPU	110.4	33.2×
<code>FixedEffectModels.jl</code>	CUDA	105.7	34.7×
<code>xhdfe</code> , Stata	CPU	89.6	41.0×
<code>xhdfe</code> , Python	CPU	85.8	42.7×
<code>xhdfe</code> , Stata	CUDA	25.9	141.4×
<code>xhdfe</code> , Python	CUDA	17.5	209.9×

Comparison with other implementations (firm-specific seniority model)

Implementation	Backend	Seconds	Speedup
<code>reghdfe</code> , Stata	CPU	14,343.0	1.0×
<code>pyfixest</code>	CPU	—	—
<code>FixedEffectModels.jl</code>	CPU	2,776.1	5.2×
<code>fixest</code>	CPU	1,370.7	10.5×
<code>xhdfe</code> , Stata	CPU	201.6	71.2×
<code>xhdfe</code> , Python	CPU	195.3	73.4×
<code>FixedEffectModels.jl</code>	CUDA	169.7	84.5×
<code>xhdfe</code> , Stata	CUDA	69.5	206.3×
<code>xhdfe</code> , Python	CUDA	60.7	236.4×

`pyfixest` does not support the heterogeneous fixed-effect slopes used in this specification.

Companion commands and applications

What does the shared numerical core make possible beyond a single regression?

The command family extends estimation into complete workflows

Estimate and partial out

xhdfe, xfepout

regression, inference, FE
recovery, and residualization

→ **xhdfeconnected, xhdfeakm** →

Worker–firm samples

connectivity, leverage, and
AKM/KSS corrections

Explain and report

xhdfegelbach

decompositions, bootstrap,
tables, and plots

A shared numerical core supports recurring empirical tasks beyond a single regression call.

The `xhdfe` suite supports AKM/KSS and Gelbach analysis

- `xhdfe` supplies the ordinary plug-in worker and firm components of the AKM model ([Abowd, Kramarz, and Margolis, 1999](#)).
- `xhdfeconnected` constructs the leave-one-out connected sample used for KSS correction ([Kline, Saggio, and Sølvesten, 2020](#)).
- `xhdfeakm` adds leverage adjustment and leave-out variance-component corrections ([Kline, Saggio, and Sølvesten, 2020](#)).
- `xhdfegelbach` decomposes coefficient movements into additive channels ([Gelbach, 2016](#)).

One decomposition spans five econometric settings

Guimarães, Portugal, and Reis (2026) apply the Gelbach decomposition to five worker–firm applications.

Using `xhdfe-xfe`, the independent implementation recovers the paper's main qualitative component rankings despite differences in the estimation samples.

Application	Coefficient of interest	Econometric setting
Gender wage gap	Gender indicator	Time-invariant regressor
Returns to schooling	Schooling	Instrumental variables
Wage cyclicalities	Lagged unemployment	Aggregate regressor and match effects
Union wage premium	Union density	Continuous exposure
Wage persistence	Lagged wage	Dynamic panel and split-panel jackknife

The five applications complete in 34 minutes

Application	Main implementation	Runtime
Gender wage gap	xhdfegelbach	3 min 35 s
Returns to schooling	xhdfe IV and projections	6 min 34 s
Wage cyclicalilty	xhdfegelbach, xhdfe	7 min 31 s
Union wage premium	xhdfegelbach, xhdfe, xfepout	11 min 26 s
Wage persistence	xhdfegelbach	5 min 20 s
Total		34 min 26 s

QP, 1986–2024; approximately 24–43 million observations per application.

Boundaries and conclusion

What is supported by the evidence, and where are the current boundaries?

Current evidence is broad, but not universal

Supported by current evidence

- multiway HDFE and heterogeneous slopes
- IV/2SLS, weights, and clustered inference
- FE recovery and worker–firm companions
- Stata, Python, and R on one core
- explicit optional CUDA

Boundaries

- not every `reghdfe` feature is covered
- unsupported combinations remain explicit
- benchmarks depend on data and hardware
- CUDA requires Linux, an NVIDIA GPU, and the CUDA toolkit
- `xhdfe-fast` uses a looser numerical tolerance by design

AI accelerated `xhdfc` without relaxing econometric validation

- 1 **Development:** AI lowered implementation and coordination costs across a multi-language research-software project.
- 2 **Estimator:** `xhdfc` preserves familiar Stata specifications and numerical agreement while moving computation to compiled CPU and CUDA backends.
- 3 **Evidence:** measured gains reach $71.2\times$ on CPU and $206.3\times$ on CUDA in the firm-specific-slope benchmark.
- 4 **Applications:** companion commands extend the same core into worker-firm and decomposition workflows.



[github.com/reisportela/
xhdfc-xfe](https://github.com/reisportela/xhdfc-xfe)

Questions and additional validation cases are welcome.

- John M. Abowd, Francis Kramarz, and David N. Margolis. High wage workers and high wage firms. *Econometrica*, 67: 251–333, 1999. doi: 10.1111/1468-0262.00020.
- Jonah B. Gelbach. When do covariates matter? And which ones, and how much? *Journal of Labor Economics*, 34: 509–543, 2016. doi: 10.1086/683668.
- Paulo Guimarães, Pedro Portugal, and Hugo Reis. O processo de triagem no mercado de trabalho português, incluindo um guia prático sobre a decomposição das estimativas dos coeficientes de regressão. *Revista de Estudos Económicos*, 12(3):1–28, 2026.
- Patrick Kline, Raffaele Saggio, and Mikkel Sølvsten. Leave-out estimation of variance components. *Econometrica*, 88: 1859–1898, 2020. doi: 10.3982/ECTA16410.