Trends in CPU Performance Improvement for Scientific and Engineering Workloads

Benjamin Whipple

Summary

Motivation. Many scientific and engineering workflows are constrained by available CPU performance, and planning future modeling capabilities requires realistic assumptions about how fast that performance is improving. Moore's law no longer provides a reliable link between device scaling and end-to-end application speed, and existing quantitative estimates of 10–20% annual improvement are largely drawn from HPC cluster contexts that may not reflect workstation-class computing.
Methods. We construct a latent measure of general CPU performance from eight benchmarks in the Phoronix Test Suite, spanning a range of single- and multi-threaded workloads. This latent scale is anchored to NumPy and AMG benchmarks for interpretability and then modeled as a function of launch date and price using a set of nested hierarchical linear models, with model selection via PSIS-LOO and parallel modeling of the constituent tests for comparison.
Outcomes. The generalized performance factor appears internally valid, reproducing the main temporal trends observed in the individual benchmarks. Within the post-2017 Ryzen era, we estimate annual CPU performance gains of roughly 17–23% for AMD and 8–15% for Intel at comparable price points, consistent overall with an improvement rate on the order of 10–20% per year but with clear brand- and price-segment–dependent variation.
Implications. Our latent-factor approach provides an empirically grounded view of CPU performance trends. The improvement rates we observe indicate that it is reasonable to expect doubling in compute performance every 4-7 years. Expectations for tractability of computational tasks as well as the useful lifetimes of computer infrastructure can be assessed in these terms.

1 Introduction

Many areas of applied modeling rely on intensive computation, and for many such tasks the scaling of computational cost with problem size is reasonably well understood. For example, the cost of a finite element analysis is often expressed in terms of the number of degrees of freedom and the order of the numerical scheme[1]. Limits on available compute frequently drive modeling choices, for instance, the coarseness of a spatial discretization in a finite element model, with the resulting discretization error or loss of resolution often quantified in advance[1]. Conversely, many engineering and scientific problems admit approximate thresholds of computational feasibility for answering specific questions. As one example, a variance-based global sensitivity analysis of a partial differential equation using Sobol' indices may require tens of thousands of forward model evaluations to achieve acceptable precision in the sensitivity estimates[2]. Problems can therefore be classified as currently intractable but potentially feasible given a specified level of computational resources. Under assumptions about future improvements in computational performance, it should be possible to estimate when such problems will become tractable. This, in turn, could inform prioritization and long-term planning in national research programs and large-scale infrastructure roadmaps.

Despite this, relatively little quantitative work has been done to characterize the rate of improvement in CPU-based computational resources, especially at fixed price points. Historically, Moore's law suggested exponential growth in transistor counts and was often informally treated as implying similar growth in computational performance, but this relationship has broken down in recent years as power and architectural constraints have decoupled CPU performance from raw device scaling[19,20]. Certain consumer-oriented benchmarks exist, such as Geekbench, from which trends can be inferred for desktop and mobile processors[3]. However, many of these benchmarks emphasize performance characteristics that are largely irrelevant to scientific and engineering workloads, such as short-lived, cache-friendly tasks or user-interface–like operations. There is therefore good reason to doubt that trends in such scores reflect trends in performance for applications like long-running PDE solvers, Monte Carlo risk analysis, or large-scale optimization running primarily on CPUs.

Some limited work by other groups suggests an improvement rate on the order of 10–20% in performance per dollar per year[4,5]. These studies, however, focus primarily on high-performance computing systems and HPC-oriented benchmarks such as LINPACK, HPL, or dense linear algebra kernels. While informative for supercomputers, they have limited relevance to the common case of modeling conducted on workstation-class machines. Moreover, the benchmarks considered are predominantly array- and throughput-heavy, and thus have limited applicability to a broader range of algorithmic workloads executed on CPUs, including branch-heavy agent-based simulations and combinatorial optimization problems[6]. As such, there remains a clear need for further investigation.

A central challenge in this context is defining a concrete notion of performance for CPU-centric workloads. A single

numeric rate of improvement is most useful for planning purposes, but not all computational tasks share the same performance characteristics, and some classes of CPU workloads advance more rapidly than others. For instance, highly vectorized floating-point throughput on many-core CPUs has improved differently from single-thread performance on irregular, branch-heavy integer workloads, and memory-bound codes often benefit less from nominal increases in clock speed or core count. Methods from latent space modeling in quantitative psychology suggest a way forward: given a representative suite of CPU benchmarks, one can attempt to infer an underlying dimension of general computational performance[7,8].

In this work, we construct a latent space model of general CPU performance using eight different benchmarks from the Phoronix Test Suite and its associated online repository[9,10]. These benchmarks were chosen to span a range of single- and multi-threaded workloads, including traditional HPC-style kernels, compiler and interpreter-heavy tasks, and branch-intensive workloads more representative of discrete-event and agent-based simulations. We fit a latent space model using EM-PCA to identify a common underlying performance dimension and then anchor this latent scale to concrete performance baselines, thereby recovering a ratio-scale interpretation in terms of multiples of a chosen reference processor. Finally, we fit a hierarchical linear model relating this latent performance measure to processor price and launch date. Using this model, we estimate annual CPU performance improvement to be on the order of 10–20%, with meaningful differences in the rate of improvement across price segments (e.g., budget versus workstation-class CPUs) and across major vendors or brands.

2 Methods

2.1 Data

We utilized data from the Phoronix Test Suite, as published on OpenBenchmarking.org, both to identify the CPU models to include and to construct our aggregate performance measure. We considered the 12 benchmarks listed in Table 3 when defining the candidate set of CPU models. As previously mentioned, these benchmarks were selected to span a wide range of performance characteristics while also being available for a broad set of CPU models. The individual benchmarks are described in greater detail in Table 3, and the coverage of CPU models across benchmarks is visualized in Figures 5 and 6.

The benchmark data encompassed 485 distinct CPU models. For each model, we collected launch price and launch year primarily from an online CPU database[11]. Launch price data were missing for most mobile CPUs, and in any case were less meaningful for our analysis because mobile CPUs were typically not user-configurable components in the same way as desktop and server processors. We therefore excluded mobile CPU models from further analysis, leaving 386 models in our final sample. A small number of benchmark entries corresponded to multiple CPU instances; for these, we applied a multiplicative adjustment to the listed launch price. We also performed limited, ad hoc supplementation of missing price and release-year values when the information was readily available from reliable sources. Summary characteristics of the final dataset are reported in Table 2.

2.2 Factor Modeling of Performance

Performance across benchmarks was first summarized using a factor-analytic approach based on principal component analysis (PCA) with an expectation-maximization (EM) algorithm to accommodate missing values. To maintain interpretability relative to the original performance scales, we then reformulated standardized scores relative to a baseline CPU and transformed the resulting principal component factor scores to align with a known percent-performance scale. Without this calibration step, the arbitrary location and scale of the principal component scores need not correspond to the scale of the original benchmark data. We proceed to describe this process further.

We reduced our consideration to the benchmarks noted as included in Table 3. This was necessary in order to ensure the choice of a sensible baseline CPU model common to all tests. From these tests, we choose the performance of the AMD r7 7700X as our baseline model. This model is a upper-mid range consumer CPU model from 2022 which launched at $399 USD. As such, it forms an easily interpretable baseline for typical CPU performance.

The benchmark data was log or -log transformed corresponding to whether the test measured throughput or duration. This transformation ensures that all data aligns with a higher-is-better standard and is on a comparable linear scale. For each benchmark, the baseline model score was then subtracted from all scores. The resulting transformed scores can be understood as a multiplicative performance factor over the baseline on their original scales.

We then applied EM-PCA, which iteratively alternates between imputing missing values from the current low-rank reconstruction and recomputing the principal components on the completed matrix, until convergence. This procedure yields orthogonal linear combinations of tests ordered by the amount of variance explained. The procedure psuedo-code is presented in Algorithm 1, which follows the class of practical EM-based PCA methods for missing data described by Ilin and Raiko[12].

The first principal component, which captures the largest share of common variation across tests, can be interpreted as a general performance factor when the corresponding loadings are predominantly positive. In such case the associated factor scores provide a single, composite performance measure that aggregates information from the full set of benchmarks while remaining interpretable as overall performance.

We then re-expressed the first principal component on an interpretable performance scale by anchoring it to a small set of benchmarks with clear substantive meaning. Specifically, using the amg and numpy benchmarks which have clear connection to single and multithreaded performance respectively, we computed the geometric mean of their transformed scores for each CPU. This aggregate provides a single summary of percent improvement over the baseline on the original scale. We regressed this anchor summary on the PC1 scores for CPUs with observed values on the reference tests and used the fitted linear relationship to rescale all PC1 scores.

The resulting calibrated latent scores preserve the ordering and relative spacing implied by the factor model, while inheriting the interpretation of scale from the geometric-mean anchor. Similar anchoring methods are common in other applications of latent variable modeling[13]. The anchored PC1 scores are used as Performance, which is primary quantity of interest in our further analysis.

2.3 Statistical Modeling

We handled missing values in key hardware covariates using multiple imputation by chained equations implemented with random forests in the `miceRanger` package in R[14,15]. We used all collected CPU descriptors (e.g., brand, process size, clock frequency, power, memory bandwidth, core count, release time, launch price, and market) to impute the remaining missing launch price observations. The `miceRanger` algorithm iteratively fits random-forest models for each incomplete variable conditional on the others and, over 10 iterations per dataset, draws replacements using predictive mean matching to maintain realistic values and respect observed marginal distributions. We generated $m = 3$ imputed versions of the dataset, which were used as the basis for subsequent modeling of performance and its determinants.

We modeled latent performance using Bayesian multilevel regression in order to characterize how overall performance evolves over time and how it scales with cost across CPU brands. At its core, the model links performance to standardized release time, standardized log launch price, and their interaction, with brand-specific random effects on the intercept and selected slopes. This structure allows brands to differ in their average performance level and in how strongly performance responds to time and price, while still borrowing strength across brands through partial pooling[16]. Log-transforming and standardizing launch price reduces skew, makes the price effect closer to linear, and places time and price on comparable scales, which in turn aids both estimation and interpretation of the interaction.

We used weakly informative $\mathcal{N}(0,5)$ priors on fixed effects to regularize the global time and price trends, Exponential(3) priors on group-level standard deviations to discourage extreme brand-specific deviations unless strongly supported by the data, and an Exponential(1) prior on the residual standard deviation. All models in the candidate set were fit in `brms` via the CmdStan backend, using four Markov chains with 4,000 iterations per chain (2,000 warmup), and convergence and mixing were assessed using standard diagnostics. Full model specifications, including the exact random-effects structures considered, are provided in Appendix E.

To choose an appropriate brand-level structure and interaction specification, we conducted a focused model comparison on the first imputed dataset. We considered five conceptually nested candidates that varied in how strongly they pool information across brands and in whether the time–by–price interaction is allowed to differ by brand, ranging from fully unpooled fixed-effects models with brand-specific intercepts and slopes to increasingly flexible hierarchical models with brand-specific random intercepts and random slopes for time, price, and their interaction. Each candidate model was fit to the first imputed dataset using the common prior structure and MCMC settings described above, and we compared their out-of-sample predictive performance using approximate leave-one-out cross-validation (PSIS–LOO) as implemented in the `loo` package[17]. For each model, we computed the expected log predictive density (elpd_{LOO}) and summarized model differences in terms of elpd contrasts and associated standard errors. This information-theoretic comparison provided a principled basis for ranking the candidate specifications and identifying a set of models with competitive predictive performance, while also balancing goodness of fit against parsimony and interpretability[17].

On the basis of this comparison, we selected one hierarchical specification as our primary analysis model and then refit this chosen model to all $m = 3$ imputed datasets using the multiple-imputation interface provided by `brms`. The resulting posterior draws were combined across imputations following Rubin's rules as implemented in `brms`[18], thereby propagating uncertainty from both imputation and parameter estimation into all subsequent inferences.

2.4 Implementation

Primary data collection and cleaning was conducted in Python 3.12 using the typical data analysis libraries. R (version 4.2) with `miceRanger` (version 1.5.0), `brms` (version 2.23), `ggplot2` (version 4.0.1), and other related tidyverse packages. All analyses were run on a M2 Pro Macbook with 32GB of memory, on which the analysis took roughly 15 minutes to run. All code associated with the analysis but not data collection is available on Github.

3 Results

3.1 Performance Measure

Our EM-PCA yielded factor loadings shown in Table 4. We note that all factor loadings are positive, which provides an indication that the first principal component correctly corresponds to a latent generalized performance variable. This factor was found to explain over 70% of the variance in the 8 considered benchmarks. We ground the scale of the variable to the geometric mean of the transformed amg and numpy test results. This transformation is described in greater depth in Appendix D.

For the purpose of informal validation, we inspect the temporal trend of our resulting performance variable in Figure 1.

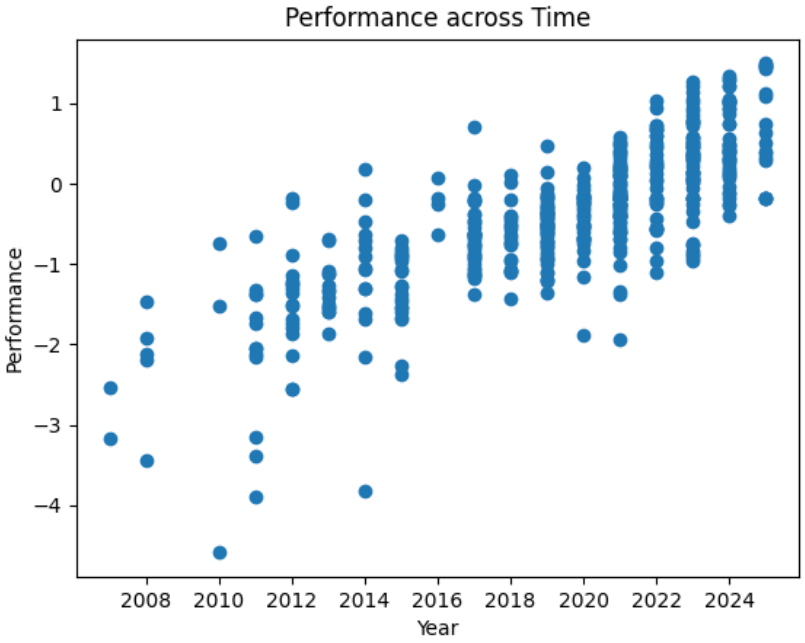

Figure 1. Visualization of the performance measure across time. Due to our scaling and anchoring, $100 \times (e^{\text{Performance}} - 1)$ can be understood to be an approximate percent improvement over the baseline CPU model AMD r7-7700X. In broad agreement with expectations, we see a broadly positive trend in the performance with time.

We can observe a positive trend in performance with time, which agrees with reasonable expectations. We note that the y-axis of Figure 1 can be interpreted as percent improvement over the AMD r7-7700X we use as baseline if transformed as $100 \times (e^{\text{Performance}} - 1)$.

The trends in Figure 1 appear broadly reasonable. Most CPUs fall below the baseline model's performance, and only a few exceed it by more than about 170% (corresponding to a value of 1 on the y-axis). Given that our baseline, the Ryzen 7 7700X, is a $400 mid–high-end consumer-grade component at launch, one might initially expect a larger number of clearly superior observations. However, roughly half of our benchmark tests are primarily single-threaded, which limits the advantage that heavily multithreaded high-end server CPUs can demonstrate in this dataset. In practice, single-threaded performance does not differ dramatically between consumer and professional-grade CPUs, so the observed distribution of relative performance is consistent with this constraint.

3.2 Model Selection

All models were fit with four chains and 8,000 iterations per chain with a 2,000 iteration warmup. Convergence diagnostics generally indicated good mixing: all monitored parameters had $\hat{R} \leq 1.01$ and effective sample sizes in the hundreds or higher, with Monte Carlo standard errors small relative to posterior uncertainties. For the fully hierarchical specifications (M3, M4, and M_full), the NUTS sampler reported a small number of divergent transitions, affecting less than 0.5% of post–warmup iterations and concentrated in regions where certain brand-level variance components approached zero. In contrast, the non-hierarchical models (M1, M2) exhibited no divergent transitions or treedepth saturations, and trace plots for these models showed no evidence of nonstationarity.

Pareto-Smoothed Importance Sampling Leave-One-Out (PSIS-LOO) model comparison shown in Table ?? indicated that the most flexible mixed-effects model M_full attained the highest elpdLOO, but the simpler alternatives M1, M2, and M4 were essentially indistinguishable from it in predictive performance, with all ΔelpdLOO ≈ -3 and standard errors of roughly 3.3–3.7. By contrast, the random-intercept-only model M3 performed clearly worse, with a substantially lower elpd and larger associated uncertainty.

Given this near tie among M_full, M1, M2, and M4, we ultimately adopted M2-the fixed-effects model with brand-specific intercepts and slopes-as our primary specification. M2 delivers competitive out-of-sample predictive accuracy while exhibiting the cleanest sampler diagnostics (no divergent transitions and excellent mixing), and we therefore use it as the main basis for inference.

3.3 Calibrated Model

We calibrate M2 on the full collection of imputed datasets. The calibrated model achieved a Bayes R^2 of $0.717(0.688, 0.740)$. The resulting fixed-effect estimates are reported in Table 1. We first interpret these parameters and then relate them to the visual summaries in Figures 2–3.

Table 1. Posterior summaries for fixed effects in model M2. Estimates are posterior means with 95% credible intervals.

Parameter	Estimate	Est. Error	2.5%	97.5%
Intercept	-0.269	0.033	-0.334	-0.204
z_{time}	0.675	0.043	0.592	0.758
$z_{\log p}$	0.208	0.033	0.143	0.273
Intel	-0.006	0.047	-0.098	0.087
$z_{\text{time}} : z_{\log p}$	0.050	0.039	-0.026	0.125
z_{time} : Intel	-0.310	0.054	-0.414	-0.204
$z_{\log p}$: Intel	0.038	0.048	-0.056	0.132
$z_{\text{time}} : z_{\log p}$: Intel	-0.136	0.051	-0.234	-0.036

In the selected fixed-effects specification (M2), standardized time and standardized log launch price both show clear positive associations with latent performance. The posterior means for z_{time} and $z_{\log p}$ are approximately 0.68 and 0.21, respectively, with relatively tight 95% credible intervals that exclude zero. This implies that, holding price fixed, later-release CPUs tend to achieve higher latent performance, and, holding time fixed, higher-priced CPUs tend to deliver better performance on the anchored scale.

The main effect for Intel relative to the reference brand (AMD) is small and highly uncertain, with a posterior mean near zero and a wide credible interval. This indicates little evidence for a systematic global intercept shift between Intel and AMD once time and price are controlled for. In contrast, the interaction between time and Intel is strongly negative: the $z_{\text{time}} \times$ Intel coefficient is clearly below zero, implying that Intel's performance gains over time are meaningfully slower than AMD's at baseline price levels. The three-way interaction $z_{\text{time}} \times z_{\log p} \times$ Intel is also negative, suggesting that this slower improvement for Intel becomes more pronounced at higher launch prices. By comparison, the remaining interactions involving log price alone ($z_{\log p} \times$ Intel and $z_{\text{time}} \times z_{\log p}$) have credible intervals that overlap zero, so the model provides only weak evidence for systematic brand differences in the pure price–performance gradient.

Interpretation of these effects is often clearer from figures. Figures 2 and 3 visualize the model-implied trajectories of performance across time and price for both brands. Consistent with the negative $z_{\text{time}} \times$ Intel and $z_{\text{time}} \times z_{\log p} \times$ Intel coefficients, these plots show AMD CPUs generally exhibiting steeper performance improvement over calendar time than Intel CPUs at comparable price points. Visual inspection of Figures 1 and 3 indicates that, at representative price levels, the model implies roughly 17–23% annual performance gains for AMD versus about 8–15% for Intel, reflecting a substantively and statistically meaningful difference in time slopes between brands.

The figures also suggest that improvement rates may vary with launch price: higher-priced AMD models appear to improve somewhat faster than lower-priced ones, while the opposite qualitative pattern seems to hold for Intel at the upper end of the price range. However, in line with the weak and uncertain price-related interaction terms in Table 1, these apparent price–by–brand differences are not strongly supported statistically and should be interpreted as suggestive rather than definitive. Overall, the parameter estimates and visualizations together tell a coherent story: both later release dates and higher launch prices are associated with better performance on average, but AMD exhibits consistently faster performance gains over time than Intel, especially toward the higher end of the price spectrum.

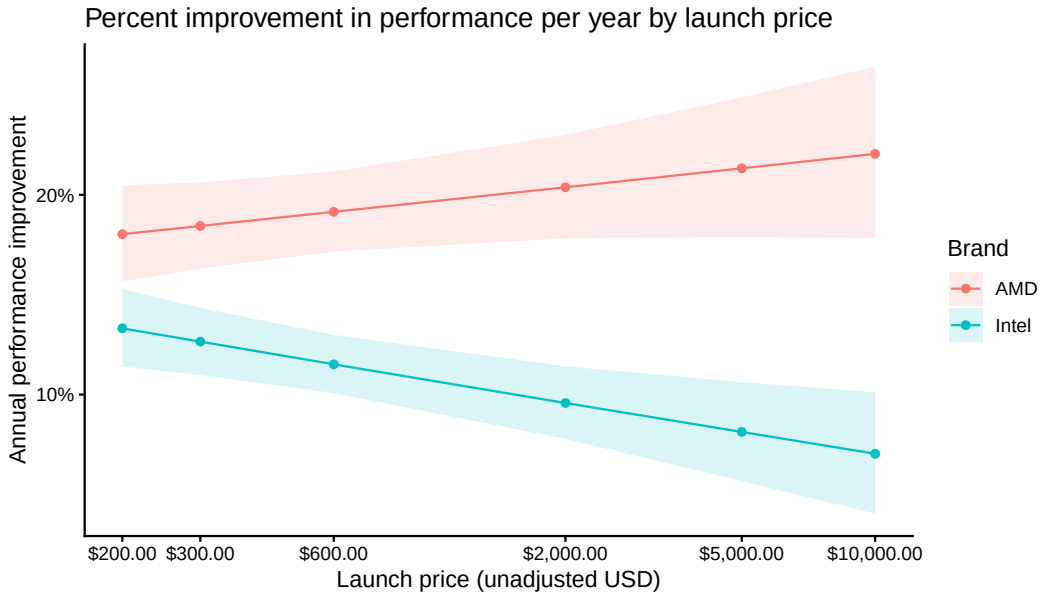

Figure 2. Percent performance improvement across launch prices and brands for CPU models. We can see practical differences in improvement rate both across launch prices and between brands.

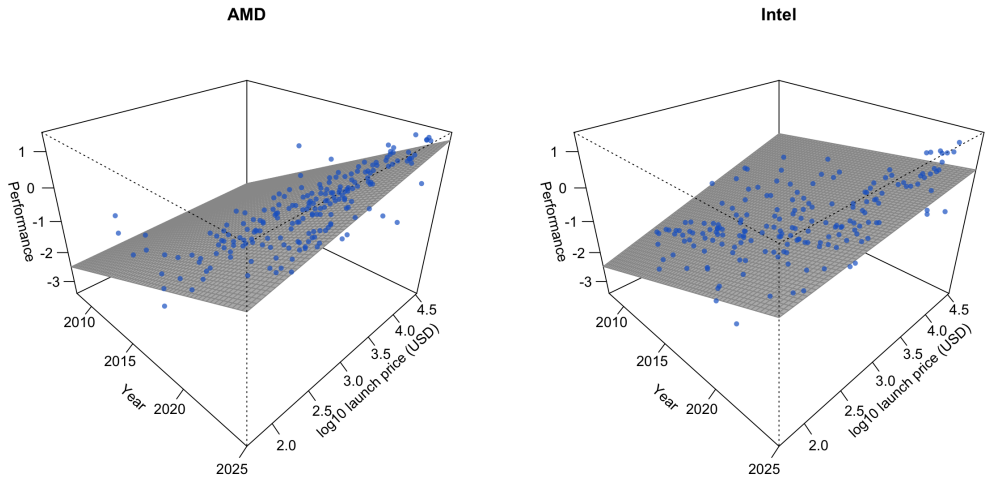

Figure 3. Surface plots of the model M2 prediction across time and log launch price. Component measurements used in calibration are shown as blue dots. The z-axis represents our generalized performance measure. The plots indicate broadly increasing performance with time and launch price and indicate differences in trends across different launch prices. Further, the magnitudes of trends appear to differ between brands.

3.4 Comparison Against Original Benchmarks

For latent space models such as we are employing, it is often useful to assess construct validity by comparing against the individual benchmarks used to construct it. For this purpose, we calibrate the M2 model on all individual tests used to construct the generalized performance measure. Trends in performance improvement rate are compared across tests against and generalized performance in Figure 4.

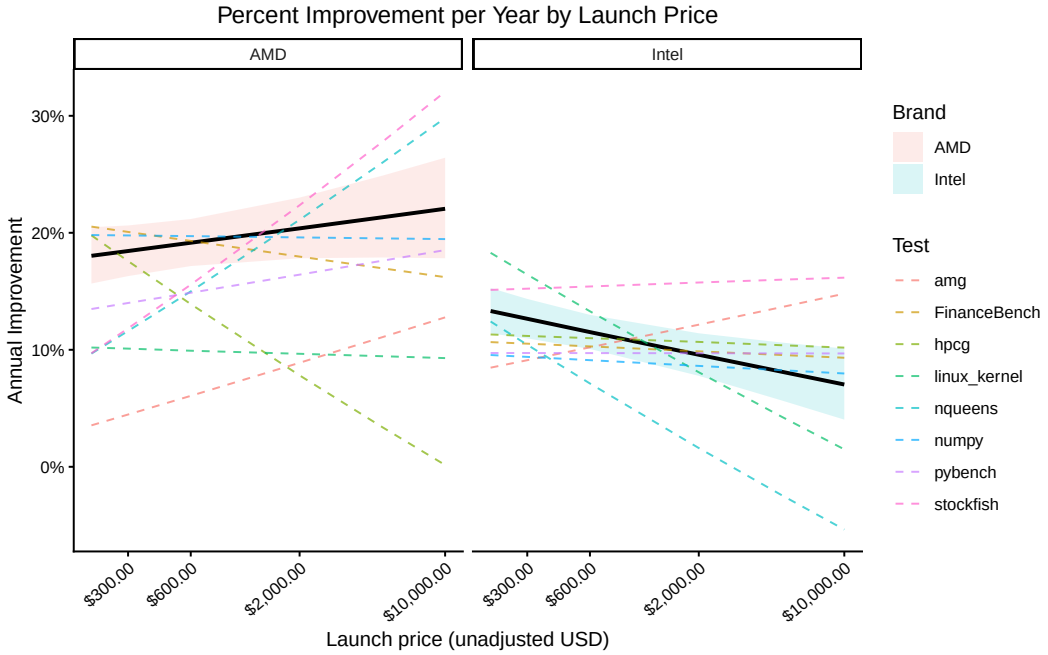

Figure 4. Percent improvement per year across launch price of CPUs as predicted by model M2. We present the results for model M2 calibrated on the generalized performance measure, as well as individual performance tests. We see broad agreement between the predicted rate of increase for the generalized performance case as for the individual tests, which further indicates the internal validity of our performance measure.

The latent space embedding can be summarized as choosing a specific weighting scheme to average trends in different test performances. As such, we look to see rough agreement in magnitude and trends with the original constituent tests in Figure 4. Overall, we assess that the behaviour and magnitude of the performance metric seems reasonable given the constituent tests. AMD models seem to be ascribed a somewhat elevated level of improvement, but it is not egregious and can reasonably be understood by the use of the numpy and amg tests for anchoring.

4 Discussion

An understanding of the rate of improvement in computational resources is crucial for assessing the feasibility of current and future research directions, especially at a policy level. Many traditional scientific and engineering tasks depend heavily on CPU performance. Historically, Moore's law was often invoked as a shorthand for exponential growth in computational capability, but in practice it describes transistor scaling rather than end-to-end application performance, and its relevance to real-world CPU performance has diminished in recent years[19,20]. Internal assessments at different organizations have proposed a range of roughly 10–20% improvement per year in general computational performance, but these studies focus on high-performance computing (HPC) cluster settings and may be less representative of the consumer and workstation environments most commonly used by scientists and engineers for modeling[4,5].

We assess that, due to the wide range of performance characteristics relevant to end users, no single benchmark can be relied upon to characterize system performance. Moreover, many composite benchmarks cover too narrow a range of CPU models to support useful predictive modeling. As such, we build a latent model of performance using eight different benchmarks to provide a broader coverage of CPU models and workload characteristics. The latent performance scale is then anchored for interpretability to the NumPy and AMG benchmarks, which are more representative of numerical scientific workloads.

We then model our generalized performance measure across time and launch price using five simple nested hierarchical linear models. Model selection by PSIS-LOO led us to select M2, a fixed-effects specification with brand-specific levels and

interactions between time and launch price. Similar modeling of the constituent performance tests was conducted using the same M2 structure for comparability.

The marginal effects of time for our generalized performance model and for the constituent tests, shown in Figure 4, provide the bulk of our findings. First, our constructed generalized performance measure appears internally valid as a representative summary of the constituent tests. Second, annual performance improvements differ systematically between AMD and Intel models and vary across CPU launch prices. Overall, we observe roughly 17–23% annual performance gains for AMD versus about 8–15% for Intel, which broadly agrees with the improvement ranges reported in prior HPC-focused work[4,5]. AMD performance improvements appear marginally higher for higher-priced models, whereas higher-priced Intel models exhibit reduced performance improvement rates.

Our analysis primarily considered models released after 2017. AMD CPU models have been increasingly dominant since the introduction of the Ryzen architecture in early 2017, whereas Intel had been dominant for some time prior to this period. Our data cannot describe these earlier trends. As such, it is important to avoid extrapolating our findings to earlier hardware generations or interpreting them as evidence about the long-run historical balance of performance between AMD and Intel CPU brands. Our results should instead be viewed as characterizing relative performance dynamics within the post-2017 Ryzen era, rather than as a definitive statement about brand dominance over the full history of x86 CPUs.

Further, our analysis is restricted to desktop and server-class hardware, for which list price data are available and relevant. Many scientists and engineers, however, make extensive use of laptops, tablets, and phones, including for computationally intensive tasks. Although one might attempt to extrapolate our findings to mobile platforms, such extrapolation would be highly unreliable without additional data and modeling.

Finally, an increasingly large set of computational methods can exploit graphics processing units (GPUs) for accelerated performance. We do not analyze trends in GPU performance on such tasks. Further work examining GPU performance across price and time, and linking GPU and CPU trends within unified models, would be informative.

Overall, our results suggest that CPU performance in the post-2017 era can be described as increasing at approximately 10–20% per year, with substantial variation by brand, price segment, and workload. Interpreted as a rough planning rule, this corresponds to a doubling of compute performance every 4–7 years at fixed launch price. Our latent-factor approach provides an empirically grounded view of these trends and, within its scope, offers a simple way to reason about when currently intractable workloads may become feasible and how long existing CPU infrastructure is likely to remain competitive.

References

1. Johnson, C. *Numerical solution of partial differential equations by the finite element method* (Courier Corporation, 2009).
2. Saltelli, A. *et al. Global sensitivity analysis: the primer* (John Wiley & Sons, 2008).
3. Primate Labs Inc. Geekbench 6. https://www.geekbench.com/ (2023). Cross-platform CPU and GPU benchmark software.
4. Norton, A., Conway, S. & Joseph, E. Price/performance predictions for the largest supercomputers in the world. Tech. Rep. HR1.0029.06.06.2019, Hyperion Research (2019). Technology Trends.
5. Panzer-Steindel, B. Cost evolution of compute and storage servers. Presentation at the 6th Scientific Computing Forum (2019). 6th Scientific Computing Forum, CERN.
6. Cormen, T. H., Leiserson, C. E., Rivest, R. L. & Stein, C. *Introduction to algorithms* (MIT press, 2022).
7. Allen, M. J. & Yen, W. M. *Introduction to measurement theory* (Waveland Press, 2001).
8. Little, T. D. *Longitudinal structural equation modeling* (Guilford Publications, 2024).
9. Larabel, M. & Tippett, M. Phoronix test suite. https://www.phoronix-test-suite.com/ (2008). Free and open-source benchmarking software with integration to OpenBenchmarking.org.
10. Larabel, M. & Tippett, M. Phoronix test suite and OpenBenchmarking.org. https://openbenchmarking.org/ (2010). Free and open-source automated benchmarking platform integrating the Phoronix Test Suite.
11. TechPowerUp. Techpowerup cpu specifications database. https://www.techpowerup.com/cpu-specs/ (2025). Online database of processor specifications.
12. Ilin, A. & Raiko, T. Practical approaches to principal component analysis in the presence of missing values. *The J. Mach. Learn. Res.* **11**, 1957–2000 (2010).
13. Ekstrand, J., Westergren, A., Årestedt, K., Hellström, A. & Hagell, P. Transformation of rasch model logits for enhanced interpretability. *BMC Med. Res. Methodol.* **22**, 332 (2022).
14. Wilson, S. *miceRanger: Multiple Imputation by Chained Equations with Random Forests* (2025). R package version 1.5.0.

15. R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria (2022). R version 4.2.0.

16. Gelman, A. & Hill, J. *Data analysis using regression and multilevel/hierarchical models* (Cambridge university press, 2007).

17. Vehtari, A., Gelman, A. & Gabry, J. Practical bayesian model evaluation using leave-one-out cross-validation and waic. *Stat. computing* **27**, 1413–1432 (2017).

18. Bürkner, P.-C. brms: An r package for bayesian multilevel models using stan. *J. statistical software* **80**, 1–28 (2017).

19. Moore, G. E. Cramming more components onto integrated circuits. *Electronics* **38**, 114–117 (1965).

20. Leiserson, C. E. *et al.* There's plenty of room at the top: What will drive computer performance after moore's law? *Science* **368**, eaam9744, DOI: 10.1126/science.aam9744 (2020).

A Data Summaries

Variable	Min	1st Qu.	Median	Mean	3rd Qu.	Max
Year	2008	2017	2021	2020	2023	2025
Launch price (USD)	49	303	604	2691	3521	17000
Brand counts: AMD = 202, Intel = 184						
Missing launch price values: 51 (NA)						

Table 2. Summary statistics for CPU launch year and launch price by brand. This summary follows after having dropped models categorized as mobile/laptop models, for which price data is not meaningfully comparable.

B Test Summaries

Test	Included	Description	Multi-thread Heavy
FinanceBench	Yes	Collection of quantitative finance kernels (e.g., Black–Scholes option pricing, Monte Carlo equity options, bond pricing, repo contracts).	No
NumPy	Yes	Python/NumPy micro-benchmark exercising general array and vectorized numerical operations in a high-level scientific computing stack.	No
HPCG	Yes	High Performance Conjugate Gradient benchmark solving sparse linear systems with conjugate gradient and multigrid-like preconditioning, designed as a modern complement to LINPACK for supercomputers.	No
OpenRadioss	No	Explicit finite element solver for dynamic event analysis (e.g., crash, impact) using large-scale structural mechanics models from the OpenRadioss benchmark suite.	No
TensorFlow Lite	No	Inference benchmark for TensorFlow Lite models targeting CPU execution, representative of small to medium deep-learning workloads on edge/embedded-style networks.	No
miniFE	No	Proxy application for unstructured implicit finite element codes, implementing a CG-based finite element solve on 3D meshes and designed to stress parallel sparse linear algebra.	Yes
OpenFOAM	No	Computational fluid dynamics (CFD) benchmark based on OpenFOAM, using an automotive aerodynamics case with unsteady, 3D flow on an unstructured mesh.	Yes
NWChem	No	High-performance computational chemistry package benchmark, solving electronic structure problems with parallel quantum-chemistry workloads on molecular systems.	Yes
SPECFEM3D	No	Seismic wave propagation code (SPECFEM3D) simulating acoustic, elastic, and coupled wavefields on 3D hexahedral meshes, representative of large-scale geophysical simulations.	Yes
AMG	Yes	Parallel algebraic multigrid (AMG) solver benchmark for sparse linear systems arising from 3D problems on unstructured grids, focused on multigrid setup and V-cycle performance.	Yes
Stockfish	Yes	Chess engine benchmark based on Stockfish, an advanced open-source, high-performance C++ chess engine used here to stress CPU integer and search performance.	Yes
N-Queens (m-queens)	Yes	Solver for the N-queens problem with multithreading support via OpenMP, evaluating the time to solve large board configurations.	Yes
Timed Linux Kernel Compilation	Yes	Measures the time required to build the Linux kernel in a default configuration, representative of large-scale C/C++ code compilation workloads.	Yes
Numenta NAB	No	Numenta Anomaly Benchmark (NAB), evaluating anomaly detection algorithms on streaming time-series data by timing a suite of detectors over real and synthetic datasets.	No
PyBench	Yes	Python micro-benchmark reporting average execution times for a variety of language features (e.g., function calls, loops, object operations), providing a rough estimate of single-threaded Python performance.	No

Table 3. Overview of CPU benchmarks drawn from the Phoronix Test Suite / OpenBenchmarking.org used in this study.

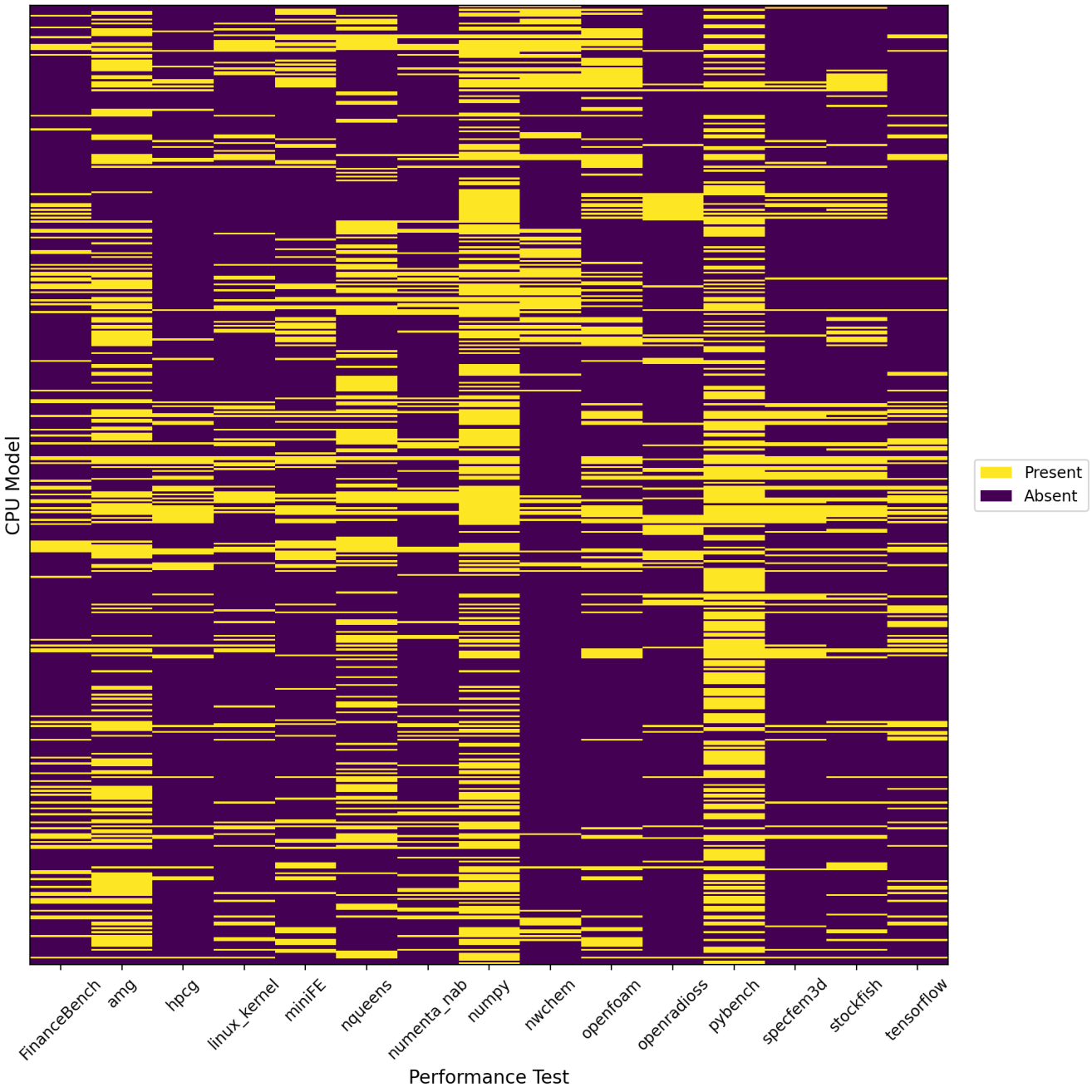

Figure 5. A heatmap visualizing the incidence of different CPU models across the different benchmarks. We note that most models have very little overlap between tests, which limits the possibility of traditional unbiased estimation methods.

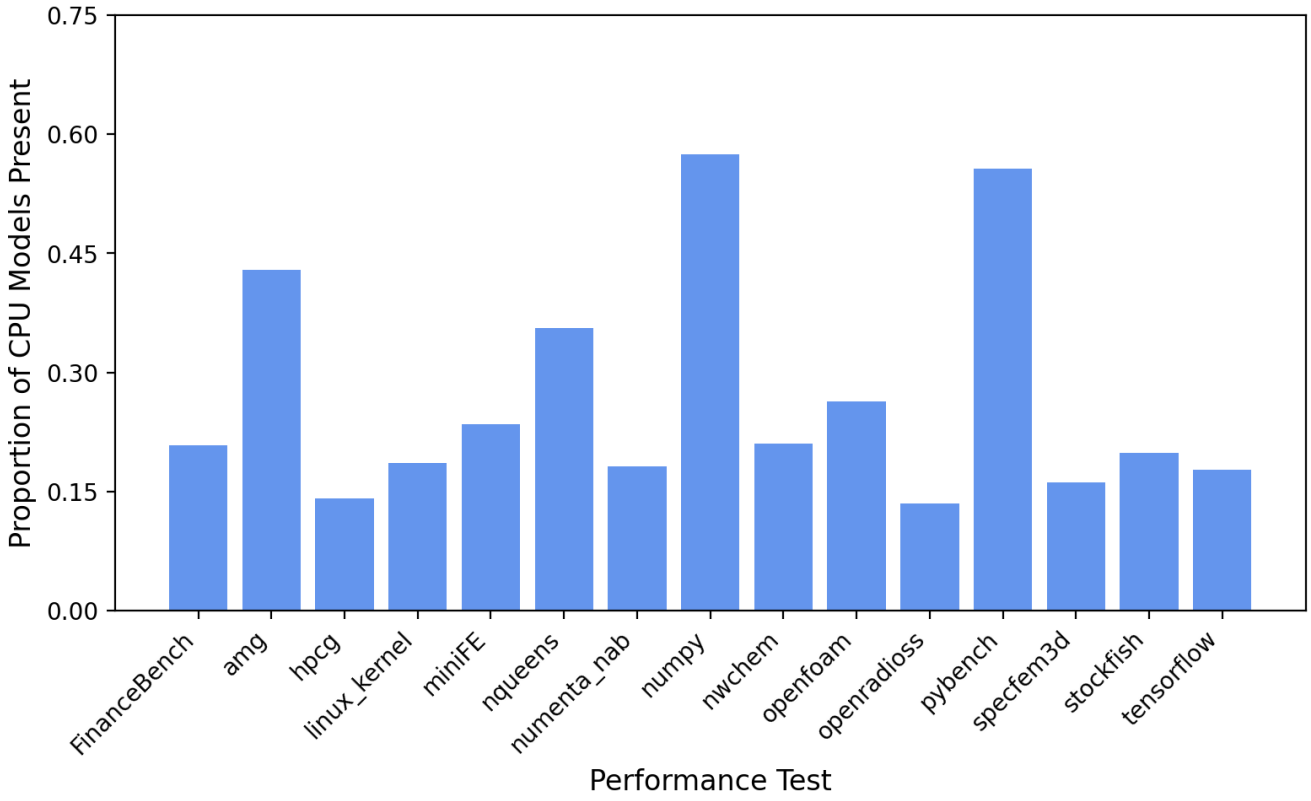

Figure 6. Visualization of the proportion of CPU models measured in each test. We can see that the four most popular tests are numpy, pybench, amg, and nqueens respectively. Most tests have only 15% of all considered CPU models measured.

C EM-PCA Procedure

Algorithm 1 EM-PCA for Incomplete Data

Require: Data matrix $X \in \mathbb{R}^{n \times p}$ with missing entries, number of components k, maximum iterations $T_{\max}$, tolerance ε
Ensure: Completed data $\hat{X}$, scores S, loadings L

Compute column means and standard deviations using observed entries of X
Standardize each column using these means and standard deviations
Define mask mask_obs indicating observed entries
Initialize $X^{(0)}$ by filling missing entries with column means
prev_err $\leftarrow \infty$
for $t = 1, \ldots, T_{\max}$ **do**
    Compute column means $m^{(t)}$ of $X^{(t-1)}$
    Center data: $X_c^{(t)} \leftarrow X^{(t-1)} - m^{(t)}$
    Perform SVD: $X_c^{(t)} = USV^\top$
    Keep first k components: $U_k \leftarrow U_{[:,1:k]}$, $S_k \leftarrow S_{1:k}$, $V_k \leftarrow V_{[:,1:k]}$
    Scores: $S^{(t)} \leftarrow U_k S_k$
    Loadings: $L^{(t)} \leftarrow V_k$
    Reconstruct: $\hat{X}^{(t)} \leftarrow S^{(t)}(L^{(t)})^\top + m^{(t)}$
    Form $X^{(t)}$ by copying observed entries from X and replacing missing entries with $\hat{X}^{(t)}$
    Compute RMSE on updated missing entries to obtain err
    if $|\text{prev\_err} - \text{err}| < \varepsilon$ **then**
        break
    end if
    prev_err $\leftarrow$ err
end for
$\hat{X} \leftarrow X^{(t)}$, $S \leftarrow S^{(t)}$, $L \leftarrow L^{(t)}$
return $\hat{X}, S, L$

D Latent Variable Mappings

The EM-PCA procedure results in the principal component loadings presented in Table 4.

Table 4. Factor loadings for the first latent performance component.

Variable	Loading
FinanceBench	0.459
amg	0.290
numpy	0.258
hpcg	0.386
stockfish	0.360
nqueens	0.360
linux_kernel	0.436
pybench	0.200

To place the latent generalized performance score on the same scale as the anchor benchmark, we fit a simple linear regression of the form

$$y_i = \alpha + \beta\, g_i + \varepsilon_i, \qquad \varepsilon_i \sim \mathcal{N}(0, \sigma^2), \tag{1}$$

where g_i denotes the latent *Generalized Performance* for observation i and y_i is the corresponding grounded (baseline-scaled) performance measure. Ordinary least squares yields

$$\hat{\alpha} = -0.3963, \tag{2}$$

$$\hat{\beta} = \ 0.2097, \tag{3}$$

so that the anchored scale is given by

$$\widehat{y_i} \ = \ -0.3963 \ + \ 0.2097\, g_i. \tag{4}$$

.

E Mixed Effects Models

Candidate models.

$$\begin{aligned}\text{M1:}\quad & y_{ij} = \alpha_{0j} + \alpha_{1j}\, z_{\text{time},ij} + \alpha_{2j}\, z_{\log p,ij} + \varepsilon_{ij},\\ & \varepsilon_{ij} \sim \mathcal{N}(0, \sigma^2),\end{aligned}$$

with a separate intercept and slopes for each brand (no partial pooling; all brand effects are treated as fixed).

$$\begin{aligned}\text{M2:}\quad & y_{ij} = \alpha_{0j} + \alpha_{1j}\, z_{\text{time},ij} + \alpha_{2j}\, z_{\log p,ij} + \alpha_{3j}\, z_{\text{time},ij} z_{\log p,ij} + \varepsilon_{ij},\\ & \varepsilon_{ij} \sim \mathcal{N}(0, \sigma^2),\end{aligned}$$

again with brand-specific intercepts and slopes (including the interaction), all treated as fixed effects.

$$\begin{aligned}\text{M3:}\quad & y_{ij} = \beta_0 + \beta_1\, z_{\text{time},ij} + \beta_2\, z_{\log p,ij} + \beta_3\, z_{\text{time},ij} z_{\log p,ij}\\ & \qquad + b_{0j} + \varepsilon_{ij},\\ & b_{0j} \sim \mathcal{N}(0, \tau_0^2), \qquad \varepsilon_{ij} \sim \mathcal{N}(0, \sigma^2),\end{aligned}$$

with a global time–price surface and only a random intercept by brand.

$$\begin{aligned}\text{M4:}\quad & y_{ij} = \beta_0 + \beta_1\, z_{\text{time},ij} + \beta_2\, z_{\log p,ij} + \beta_3\, z_{\text{time},ij} z_{\log p,ij}\\ & \qquad + b_{0j} + b_{1j}\, z_{\text{time},ij} + b_{2j}\, z_{\log p,ij} + \varepsilon_{ij},\\ & \begin{pmatrix} b_{0j} \\ b_{1j} \\ b_{2j} \end{pmatrix} \sim \mathcal{N}\left(\begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}, \Sigma_b \right), \qquad \varepsilon_{ij} \sim \mathcal{N}(0, \sigma^2),\end{aligned}$$

allowing brand-specific intercepts and main-effect slopes for time and price via partial pooling.

$$\begin{aligned}\text{M\_full:}\quad & y_{ij} = \beta_0 + \beta_1\, z_{\text{time},ij} + \beta_2\, z_{\log p,ij} + \beta_3\, z_{\text{time},ij} z_{\log p,ij}\\ & \qquad + b_{0j} + b_{1j}\, z_{\text{time},ij} + b_{2j}\, z_{\log p,ij} + b_{3j}\, z_{\text{time},ij} z_{\log p,ij} + \varepsilon_{ij},\\ & \begin{pmatrix} b_{0j} \\ b_{1j} \\ b_{2j} \\ b_{3j} \end{pmatrix} \sim \mathcal{N}\left(\begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \Sigma_b \right), \qquad \varepsilon_{ij} \sim \mathcal{N}(0, \sigma^2),\end{aligned}$$

which is the most flexible hierarchical model, with brand-specific intercepts and brand-specific slopes for time, price, and their interaction.

LOO model comparison. Approximate leave-one-out cross-validation (PSIS–LOO) yields the following differences in expected log predictive density (elpd_{LOO}) relative to the best model (M_full):

Model	$\Delta\text{elpd}_{\text{LOO}}$	$\text{SE}(\Delta\text{elpd}_{\text{LOO}})$
M_full	0.0	0.0
M1	-2.9	3.7
M2	-2.9	3.7
M4	-3.1	3.3
M3	-15.4	7.3

Here, M_full attains the highest elpd_{LOO}, but M1, M2, and M4 differ from it by only about 3 elpd units with standard errors of roughly 3.5–3.9, indicating that these three simpler models are statistically competitive in terms of predictive performance. In contrast, M3 shows a substantially worse fit (about 15 elpd units below M_full, roughly two standard errors), and can be reasonably discarded. Consequently, model choice among M_full, M1, M2, and M4 can be based on a trade-off between flexibility and interpretability, without strong evidence that further increases in complexity would yield meaningful predictive gains.