Emotional Appeals as Drivers of Social Media and Advertising Engagement in Real-World Marketplaces: Using AI to Code Variables in Consumer Research

Anirban Mukherjee
Hannah Hanwen Chang

27 February, 2025

Anirban Mukherjee (anirban@avyayamholdings.com) is Principal at Avyayam Holdings. Hannah H. Chang (hannahchang@smu.edu.sg; corresponding author) is Associate Professor of Marketing at the Lee Kong Chian School of Business, Singapore Management University. This research was supported by the Ministry of Education (MOE), Singapore, under its Academic Research Fund (AcRF) Tier 2 Grant, No. MOE-T2EP40221-0008.

Abstract

Marketers often employ emotional appeals to evoke specific feelings such as happiness, excitement, or nostalgia, thereby fostering a positive connection between consumers and brands. However, the efficacy of emotional appeals in driving consumer engagement in the real-world remains unclear. While field data is readily available, the study of emotional appeals at scale is constrained by the high costs of coding data manually. In this research, we propose the use of AI systems to algorithmically code emotional appeals in large-scale, unstructured marketing data. We present novel methodology that leverages state-of-the-art automatic machine learning to develop custom AI systems. We show that directly including AI-coded variables in econometric models can lead to inconsistent estimates. We develop novel estimators based on measuring moment matrices in a calibration subsample of the data. We apply our approach to both synthetic data and real-world field data, examining the effectiveness of humor and physical appeals on Instagram and YouTube. Our methodology is computationally stable, consistent, and highly efficient. Results show that our proposed estimators effectively bridge the gap between highly precise but smaller scale manual coding, and more imprecise but large-scale AI-based coding.

Keywords: Emotional Appeals, Humor, Physical Appeal, Social Media, Advertising, Artificial Intelligence.

JEL codes: C18, C55, C81, M31, M37.

Contents

1 Introduction

"... the best assurance that you have really discovered something is if you can show that it replicates in meaningful task environments."

—Editorial, *Journal of Marketing Research* (Meyer 2015)

"... lab experiments benefit from the complementarity of field studies in which participants are making real choices."

—Editorial, *Journal of Marketing Research* (Hamilton 2024)

Emotional appeals are ubiquitous in marketing. Humor, for instance, has been extensively studied in controlled laboratory settings across disciplines such as communications, economics, marketing, and psychology (e.g., Eisend 2009; Gulas and Weinberger 2006). These studies consistently demonstrate that humor plays a pivotal role in advertising, transcending mere entertainment to serve as a powerful catalyst for emotional engagement and foster positive brand associations (Chattopadhyay and Basu 1990; Warren, Barsky, and McGraw 2018). By humanizing brands, humor creates a sense of relatability and trust among target audiences (Beard 2005), while its strategic use significantly improves brand recall—making it an essential component of effective marketing campaigns (Chung and Zhao 2003).

Similarly, physical appeal has been widely examined in laboratory settings, where it has been shown to enhance ad engagement, as reflected in increased attention, positive affect, and purchase intentions (Dahl, Sengupta, and Vohs 2009; Sengupta and Dahl 2008; Shanks et al. 2015). Physical appeal serves to attract and hold viewers' interest, creating an affective association with the brand (Ordabayeva, Lisjak, and Jones 2022), and its effects have been observed across various media, including print, television, and online advertising (Reichert 2002). Existing research underscores the importance of excitement—a high-arousal and positively valenced emotional reaction, often a precursor to engagement (Lang and Bradley 2010)—as a crucial emotional response evoked by advertisements that manifest humor and physical appeal (e.g., Olney, Holbrook, and Batra 1991; Pham, Geuens, and De Pelsmacker 2013).

However, despite extensive laboratory research, the efficacy of emotional appeals in real-world marketplaces remains uncertain. While field data is readily available, the challenge of studying emotional appeals at scale is, among other factors, compounded by the high costs of manually coding large datasets. As a result, much of the existing evidence derives from controlled laboratory studies, which may not fully capture the complexities and nuances of real-world environments. For instance, Eisend (2009), in a meta-analysis of humor in advertising, notes that "studies, which are mostly performed in controlled laboratory settings, are only mildly amusing," and their effects "may therefore differ from the effects of real-world advertisements" (also see Woltman Elpers, Mukherjee, and Hoyer 2004). Similarly, Lull and Bushman (2015), in a meta-analysis of sexual and violent media content, highlight the difficulty of measuring real-world purchase behavior. They note that "absent purchase behavior, we are left with measures that are easier to collect in laboratory experiments," which may not fully reflect real-world consumer responses.

Yet recent findings underscore that emotional appeals matter deeply in live consumption settings, reinforcing the need to move beyond laboratory data. For instance, Garcia-Rada, Norton, and Ratner (2024) provides emerging evidence that consumers' desire for shared emotional experiences—such as creating lasting memories—fundamentally shapes real-world consumption choices. Similarly, Wu and Morwitz (2025) shows that when consumers articulate both emotional and rational aspects in online reviews, they experience affective recovery and are more likely to repurchase. These studies not only confirm the critical role of emotional expression in consumer behavior but also affirm the need to examine "consumer behavior that occurs within the milieu of actual marketplace phenomena" (Shimp 1994). Indeed, as the quotes at the beginning of our article illustrate, bridging the gap between controlled experiments and real-world generalizability has long been a challenge in consumer research (Alba 1999; Lynch Jr 1982; Simonson et al. 2001).

A major obstacle to studying constructs such as emotional appeals in real-world marketplaces stems from the unstructured nature of the data (Lamberton and Stephen 2016, p. 162). Advertisements, social media images, and videos often contain emotional appeals, but these appeals are not labeled or classified within the data itself. Researchers observe structured outcome variables,

such as the number of likes, video play duration, and sales revenue. They may be interested in mapping these structured outcome variables (e.g., likes) to variables embedded in the unstructured data (e.g., humor) to establish causal relationships (e.g., Do humorous videos receive more likes?). However, because the key independent variables (such as humor) are unobserved, analyzing such relationships traditionally requires (human) research assistants to manually code the presence and extent of emotional appeals. At scale, this need for manual coding presents a significant hurdle, hindering comprehensive analyses of the efficacy of emotional appeals in real-world settings.

In this paper, we investigate the use of Artificial Intelligence (AI) systems to address this limitation. Our approach focuses on causal inference to advance theory development. We situate our analyses within the rapidly expanding and increasingly complex landscape of unstructured marketing data—social media posts, consumer reviews, and online advertising—that have long provided rich insights into consumer preferences, sentiments, and behavior, and where emotional appeals can have significant substantive implications.

We make the following key contributions. We address the question: Can AI code consumer research-relevant variables, such as emotional appeals, as effectively as human coders? We examine how state-of-the-art and generally applicable AI modeling methodologies can be used to construct variables tracking higher-order constructs such as humor, without requiring specialized hardware or sophisticated programming. To this end, we establish a practicable algorithmic data coding method and delineate its properties.

Specifically, we demonstrate the effectiveness of Automatic Machine Learning (AutoML) in developing practical, custom AI models for coding consumer behavior-relevant variables, such as emotional appeals, in social media data from Instagram and YouTube. AutoML automates the process of AI model development by enabling AI systems to discover optimal architectures for specific tasks within target datasets. This involves optimizing a meta-model that evaluates AI model performance across different configurations, yielding stable models suitable for large-scale deployment. By transforming traditionally resource-intensive tasks—such as data preprocessing, model selection, and hyperparameter tuning—into an automated workflow, AutoML makes

advanced model development more accessible to researchers without extensive expertise in AI architecture design.

Next, we examine how AI-coded variables can be integrated into econometric models. Two sources of uncertainty caution against the uncritical use of AI-coded variables. First, the unstructured data itself may be systematically less informative than structured data. For example, contrast structured data from traditional vendors, such as J.D. Power's survey of car owners, with unstructured data from social media posts or online forums. While social media discussions can offer complementary, rich, and candid insights, they often lack the structured, well-defined variables found in traditional surveys. A consumer satisfaction survey, like J.D. Power's, might include structured ratings on aspects of the service or product experience, such as 'dependability' or 'quality.' While social media posts or forum discussions may also touch on these dimensions, they are likely expressed in natural language, with only some discussions directly relevant to these aspects. Thus, large-scale measures derived from such unstructured data may be intrinsically noisy.

Second, an AI system functions by constructing an internal numeric description of the data (an embedding). As AI embeddings derive from data features, and AI systems are statistical models, a divergence between the AI's coding and human judgments (i.e., 'AI-coding error') is likely. This AI-coding error is likely to be correlated with the data features either incorporated directly as variables into the econometric model, or correlated with other variables that are correlated with the data features. This in turn leads to an omitted variable bias (Grewal and Orhun 2024) in a naïve plug-in estimator that employs the AI-coded variables directly.

Crucially, as we show, the implications of this bias are *a priori* unclear: it may alternately attenuate or amplify estimates, change the directionality of estimates, and inflate standard errors. These issues can lead to *both* Type I (false positive) and Type II (false negative) errors in hypothesis testing, resulting in situations where *spurious* causal relationships are *inferred* or *true* causal relationships are *overlooked*. Moreover, as we show in our empirical studies, even when the AI-coded variables are correlated with the human-coded variables at 0.97—an extremely high

correlation—and the data is large-scale, the bias engendered by AI-coding error *persists*. This is because the bias is not a result of orthogonal errors but of a systematic dependence of the coding error on the incorporated independent variables.

We establish novel estimators to address these concerns. Specifically, we propose employing a calibration subsample of the data to infer key statistical properties (specifically, moment matrices) of the data-generating process. This enables us to model the statistical dependence between AI-coding errors and independent variables. We derive new closed-form estimators that are both computationally efficient and straightforward to implement—two features critical to the successful application of a method in large-scale data analysis. To facilitate implementation, we provide both a guide to illustrate use of our methodology and detailed, commented R and Python code that can be adapted to other research questions, in the web appendices.

To validate our proposed estimators, we conduct Monte Carlo simulations to assess their robustness and accuracy in a controlled environment before applying them to real-world data. We compare effect sizes obtained through three approaches: (1) analysis using only manually-coded measurements, (2) direct use of AI-coded variables in econometric models without error correction, and (3) incorporation of our proposed bias-corrected estimators. Results indicate that naïve plug-in estimators can yield estimates that differ significantly (i.e. at $p < 0.95$) in *both* direction and magnitude from ground truth, and lead to substantively *inaccurate conclusions*. Our proposed estimators bridge this gap, providing consistent estimates while capitalizing on the scalability of AI coding to large sample sizes.

We further validate our approach in two empirical studies using large-scale real-world datasets: Instagram influencer posts and YouTube video advertisements. To rigorously evaluate AI-coded emotional appeals as proxies for human judgments, we manually code the entire dataset to establish ground truth—an effort that is typically infeasible but made possible here through dedicated research funding. This allows us to directly compare AI-coded variables with human-coded benchmarks and assess the performance of our estimators. The results align with our simulations, demonstrating that uncorrected AI coding can lead to substantively different conclusions, whereas

our bias-corrected estimators yield reliable inference. Together, these analyses underscore AI's potential as a scalable coding tool in consumer research, highlight the econometric implications of AI-coding errors, and support our proposed econometric adjustments.

We organize our article as follows: First, we synthesize a framework for the use of AI to code consumer behavior relevant variables—the key innovation of our paper. We describe imprecision in AI-coded variables and its implications for marketing models, considering potential correlations of errors with independent and instrumental variables. We develop novel econometric estimators and demonstrate their application in various contexts. Detailed derivations and descriptions of the estimators are provided in the web appendices; in the manuscript, we present analytical and empirical evidence of the estimators' viability, accuracy, and precision, including a detailed set of simulation studies set within a consumer behavior research context. We then describe the AI-coded variables, the presence of AI-coding imprecision, its implications for estimates, and the efficacy of our proposed approach in two real-world applications: Instagram posts and video advertisements. We close by noting the wide-ranging relevance of our methodology. Our proposed framework lays the groundwork for future investigations into the impact of emotional appeals and other consumer research-relevant variables in large-scale, real-world data environments.

2 Methodological Framework

We begin with the premise that although laboratory studies have identified a range of theoretically relevant variables—including emotional appeals—that influence consumer behavior (e.g., purchase decisions, brand recall, emotional connections, engagement, and overall satisfaction), extending the investigation of these variables to large-scale, real-world settings has proven challenging, in part because field data are unstructured. Researchers have access to vast, relevant—but unstructured—datasets such as product descriptions, consumer reviews, social media posts, and advertisements. Yet, coding these data for consumer behavior–relevant attributes (e.g., quantifying the degree of humor in an advertisement) is both time-consuming and prohibitively expensive at scale, thus

limiting the feasibility of studies (Wedel and Kannan 2016).

Moreover, there is compelling reason to examine these variables at scale. For example, when studying the effects of humor on video advertising engagement, several critical questions arise. How do these effects vary across cultures? Humor is deeply nuanced; an advertisement perceived as hilarious in one culture might be ineffective—or even offensive—in another. For instance, Pepsi's 2017 advertisement featuring Kendall Jenner was widely criticized for trivializing social justice movements, highlighting the risks of humor that fails to resonate with diverse audiences. Empirical studies have demonstrated cultural differences in humor perception, underscoring the importance of context in advertising effectiveness (Alden, Hoyer, and Lee 1993).

Furthermore, does the effectiveness of humor differ across product categories? Research suggests that humor may work better for low-involvement products, where consumers are more receptive to lighthearted messaging, than for high-involvement products, where detailed information is prioritized (Weinberger et al. 1995). And does the impact of humor vary depending on the specific engagement metric? Evidence suggests that it might. For example, in a large-scale study of digital advertising, Tucker (2015) found that while humorous content boosted social interactions (e.g., shares and discussions), it did not necessarily increase watch times or enhance cognitive processing—particularly for primarily informational ads.

Addressing these questions requires datasets that span diverse cultural contexts, product categories, and engagement metrics—a scope often unattainable in controlled laboratory settings. Although laboratory studies are invaluable for uncovering the underlying processes by which attitudes form and change, they rarely capture the many contextual factors that shape real-world consumer responses. Petty, Wegener, and Fabrigar (1997), for instance, highlight how persuasion models developed in controlled environments—such as the Elaboration Likelihood Model—can overlook factors like competing messaging, social pressures, or cultural norms present outside the lab. Thus, emotional appeals that appear effective under controlled conditions may not always translate seamlessly to naturalistic settings.

Critically, even in Randomized Controlled Trials (RCTs) in real-world environments, the need

for scalable measurement remains. For instance, when running a real-world RCT in which subsets of users are exposed to more or less humorous advertising, the researcher must still gauge how strongly each ad actually conveys humor. In such a study, it would be impractical to rely on human assistants to manually code thousands—or even millions—of data points (e.g., advertisements). Therefore, in such scenarios, automated coding becomes indispensable.

Prior research supports the importance of large-scale RCTs. For instance, Gordon et al. (2019) and Zantedeschi, Feit, and Bradlow (2017) critique the validity of observational data-based advertising studies, advocating for large-scale field experiments to better capture real-world effects. Lee et al. (2013) analyze over 100,000 Facebook posts across 800 companies and find that persuasive content, such as emotional and philanthropic appeals, significantly increases user engagement. In contrast, purely informative content—such as mentions of prices or product features—reduces engagement when used in isolation but enhances engagement when combined with persuasive elements. These findings highlight the importance of analyzing real-world data, such as social media interactions, to design advertising strategies that resonate with nuanced consumer perceptions and behaviors.

Finally, data availability and privacy concerns further reinforce the necessity of local, automated coding methods for emotional appeals. For example, in the context of mobile advertising, privacy regulations may prohibit the recording and transmission of the actual advertisements viewed by a consumer. As a result, the data may include only an anonymized user identifier, a few descriptors such as the app used to view the content, and the consumer's response, but not the advertisement itself. In such cases, automated coding may be feasible if the advertisement is coded locally on-device (e.g., humor is coded locally on a user's mobile phone), and only the coded data—and not the raw data—are transmitted to the advertiser. This approach ensures data privacy while still enabling analysis essential for targeting and positioning.

2.1 Proposed AI-Facilitated Marketing Research Process

To address the challenges of analyzing unstructured marketing data at scale, we present a five-step framework that integrates manual coding of a calibration subsample with large-scale AI-based analysis.

1. **Step 1: Obtain Unstructured and Structured Data:** Collect raw, unstructured data—such as social media posts, consumer reviews, product images, or video content—along with any associated structured data (e.g., likes, sales figures). The unstructured data can then be linked to the structured data to support hypotheses or used as stimuli in RCTs.

2. **Step 2: Human-Code a Calibration Subsample:** Manually code a representative subset of the unstructured data to establish a calibration subsample–a subsample of the data with "ground truth" for key constructs (e.g., emotional appeals like nostalgia, humor, or physical appeal). This calibration subsample serves two critical functions: it is used both to train the AI model and to estimate the statistical properties of the AI-coding errors.

3. **Step 3: AI-Code the Complete Dataset (Using AutoML):** If no suitable pre-trained AI model exists, leverage AutoML to train a custom AI model. Deploy this model (or the pre-trained model) to code the *entire* dataset, including the calibration subsample. AutoML automates tasks such as data preprocessing, feature extraction, model selection, and hyperparameter tuning, making AI deployment accessible to researchers without deep technical expertise. Moreover, using pre-trained components (e.g., encoders) enhances efficiency. (See Web Appendix § B for an implementation guide on using AutoML to code nostalgia.)

4. **Step 4: Estimate Moment Matrices in the Calibration Subsample:** Using both the manually coded and AI-coded values from the calibration subsample, estimate key statistical properties—specifically, the relevant cross-product matrices and covariances—that capture the relationship between the AI-coding errors (i.e., the differences between human-coded

"true" values and AI predictions) and the independent variables (including any controls or instruments). This step is essential for identifying and correcting biases in inferences. For example, if AI-coding errors are correlated with factors like image resolution or audio quality, these relationships can be explicitly modeled to ensure bias-free (consistent) inference.

5. **Step 5: Estimate the Econometric Model in the Complete Dataset:** Integrate the AI-coded variables (for all observations) and the moment matrices (estimated in Step 4) into the estimation routine. Apply our bias-corrected estimators (detailed in Web Appendices § C and § D) to adjust for AI-coding errors. This approach is applicable to both single-equation models (e.g., OLS, 2SLS) and system-of-equations models (e.g., 3SLS). Use bootstrap procedures to derive valid standard errors. Methods for handling manual-coding errors are also provided (Web Appendix § E). This final step bridges the gap between large-scale but potentially imprecise AI coding and small-scale but precise manual coding, enabling robust causal inference.

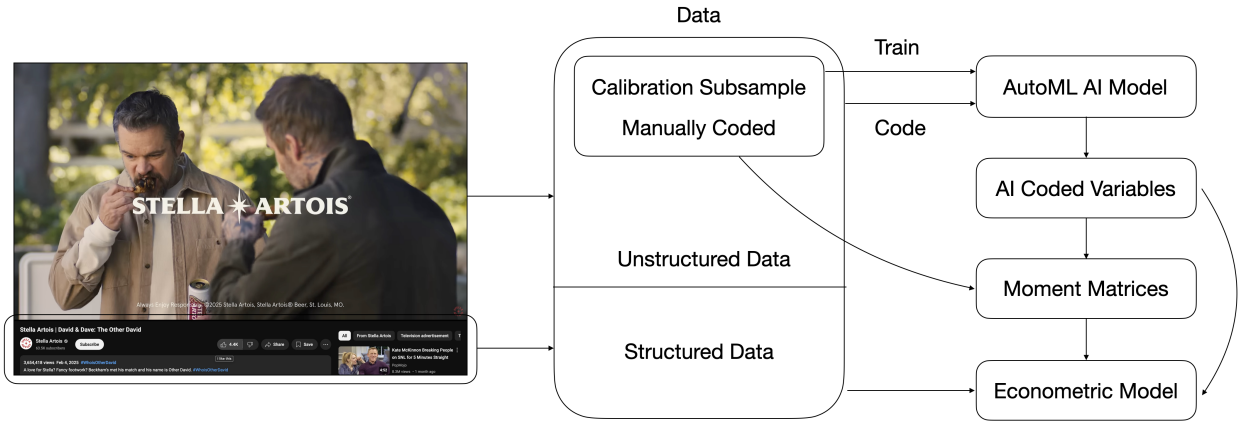


Figure 1: General Workflow in AI-Facilitated Marketing Research

Note. The left side shows a YouTube page consisting of unstructured data (the video) and structured variables (e.g., brand, likes, views). A subset of these videos is manually coded for the focal variable (e.g., humor). The resulting labels train the AutoML AI model, which then codes the complete data. The difference between AI predictions and the manually coded values in the calibration subsample is used to form error-correction moment matrices. These matrices and the structured data are then fed into an econometric model (e.g., estimating the effect of humor on likes), producing bias-corrected inference.

Figure 1 illustrates our proposed framework for AI-facilitated marketing research. On the left is the YouTube page of a recent advertisement by Stella Artois. At the top is the video

advertisement itself, representing the unstructured data that must be coded. Below are several key pieces of structured information, including the channel hosting the advertisement (here, the brand being advertised), the number of likes the video has received, and the number of views. In this example, we treat these variables as structured information; in other studies, the data might include additional independent and dependent variables.

The right side of the figure illustrates the workflow. First, both the video advertisement (unstructured data) and the structured information are recorded (Step 1). A subset of these video advertisements is then selected as a calibration subsample and manually coded for the focal independent variable—for instance, humor (Step 2). The manually coded measures in the calibration subsample train the AutoML AI model, which then codes the remaining video advertisements, thereby scaling the manual-coding process (Step 3). The difference between the AI's predictions and the manually coded values in the calibration subsample is used to compute the error-correction moment matrices, following formulas provided in the manuscript and Web Appendix § C, D, and E (Step 4). Finally, the structured data, the AI-coded measures, and the moment matrices are used in an econometric model in which the dependent variable of interest (e.g., number of likes) is regressed on both the other structured independent variables (e.g., the advertised brand) and the AI-coded variables (Step 5).

To facilitate implementation, we provide a detailed implementation guide in Web Appendix § B that demonstrates this workflow in the context of a hypothetical research study that mirrors the structure of our simulation studies. It walks through each step of the process, offering practical insights and recommendations, along with a detailed and thoroughly commented Python implementation. We also provide detailed simulation code written in R that can be adapted to other research questions.

3 Method Development: AutoML and Bias-Corrected Estimation

We now detail our key methodological innovations, which begin with Step 3. Steps 1 and 2 (obtaining unstructured or structured data and manually coding a calibration subsample, respectively) are standard procedures; we therefore proceed directly to our novel contributions. We first discuss using AutoML to build custom AI models for coding emotional appeals (Step 3). We then discuss estimating moment matrices in the calibration data (Step 4) and integrating these components for inference in the complete dataset (Step 5). Here, we present the estimators for single-equation and system-of-equations models (Step 5), with their derivation provided in Web Appendix § C and § D, respectively. Finally, we address coding error in the manually coded calibration subsample, with a detailed derivation in Web Appendix § E.

3.1 Step 3: AutoML

In many marketing research applications—such as detecting brand logos in social media images—existing AI models may suffice for coding the data, provided that both the raw data format and the target variable have been investigated in prior work. In such cases, researchers can directly apply models trained in previous studies or provided by commercial sources. However, when the focal construct (e.g., humor) or data format (e.g., video feeds) is novel to the machine learning literature—and when an appropriate pre-trained model is unavailable—a custom AI model must be built.

We advocate an *AutoML* approach in these situations. AutoML is a methodology that autonomously searches for optimal model architectures and configurations (He, Zhao, and Chu 2021; Hutter, Kotthoff, and Vanschoren 2019). By automating complex iterative tasks—such as data preprocessing, feature engineering, and hyperparameter tuning—AutoML lowers both the expertise and resource requirements needed to develop advanced ML solutions. Consequently, it

is particularly well-suited for applied marketing researchers seeking to code variables at scale without expending substantial effort on specialist programming.

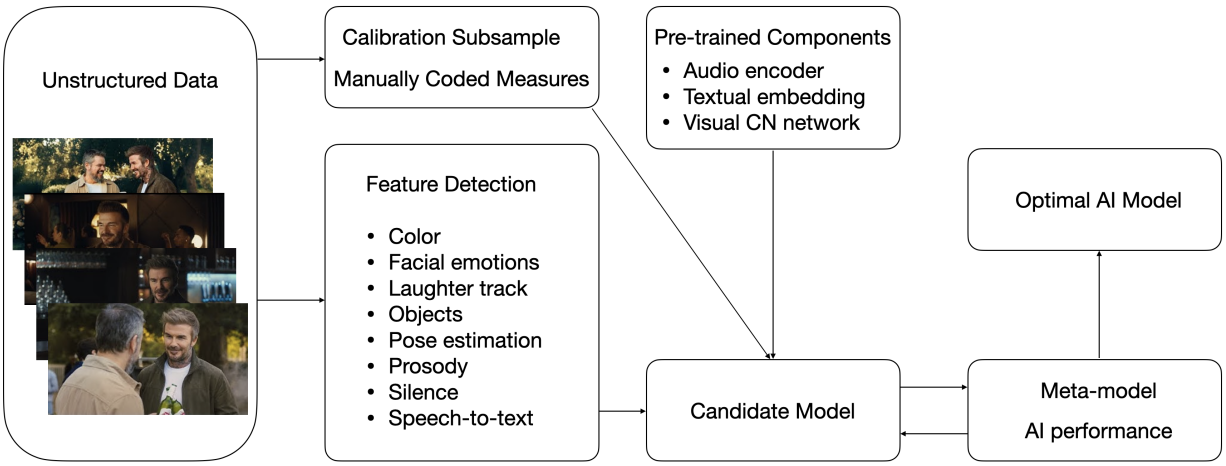


Figure 2: General AutoML Workflow for Detecting Humor in Video Advertisements

Note. A calibration subsample of video advertisements is manually coded for the focal variable (e.g., humor level). The videos are then processed via feature-extraction modules (e.g., laughter-track detection, facial-expression recognition, or pretrained models) to generate inputs for candidate AI architectures, which are systematically searched by a meta-model. The resulting best-performing solution is subsequently applied to code humor in the original (and potentially much larger) set of unstructured videos.

Figure 2 illustrates a typical AutoML pipeline for detecting humor in unstructured videos. Each ad is first manually coded on a 7-point humor scale in a calibration subsample. The next step involves extracting relevant features—such as scene boundaries, facial expressions, audio segments, or pretrained embeddings. A meta-model then evaluates candidate configurations that vary in hyperparameters, model architectures, or feature sets (Feurer et al. 2015; Vilalta and Drissi 2002). By systematically comparing performance across these configurations, the AutoML system arrives at an optimal model, which can then code humor for all remaining videos. Table 1 provides examples of features that may help identify humor in advertisements.

In our implementation guide (Web Appendix §B), we demonstrate this process with a concrete example of coding nostalgia appeals in advertisements. We show how a manually coded calibration subsample can train an AutoML model to predict nostalgia intensities, and how these model predictions can integrate with engagement data when estimating econometric relationships. Detailed Python code—using libraries such as Keras Tuner and AutoKeras—illustrates how to

Table 1: Illustrative Features for Humor Detection in Video Advertisements

Feature	Description
Color	Bright or contrasting colors signaling slapstick or playful content.
Facial Emotions	Recognizing smiles, raised eyebrows, or other facial indicators of amusement.
Laughter Track	Detecting audience laughter or canned laugh tracks in the audio track.
Objects	Identifying distinctive props (e.g., oversized items) often used in comedic setups.
Pose Estimation	Capturing exaggerated gestures or motions indicative of physical humor.
Prosody	Tracking variations in vocal rhythm, pitch, and pacing reflective of comedic timing.
Silence	Identifying pauses that reinforce punchlines or comedic beats.
Speech-to-Text	Recognizing puns, wordplay, or irony from transcribed dialogue.

implement both the AutoML workflow and the subsequent bias-corrected estimation. By following this procedure, researchers can efficiently transform unstructured marketing data into theory-relevant variables without requiring extensive machine learning expertise.

3.2 Steps 4 and 5: Estimation of the Econometric Model and Inference

We categorize the independent variables into two groups: z, which represents variables extracted from unstructured data (e.g., measures of emotional appeals), and x, which consists of observed, structured data (e.g., the number of likes in a social media study).

We consider regression models commonly used in both laboratory and field studies. The general form of our regression model is:

$$y_i = \alpha + \sum_{m=1}^{M} \beta_m z_{mi} + \sum_{l=1}^{L} \gamma_l x_{li} + \varepsilon_i. \tag{1}$$

where i indexes individual observations; y_i is the dependent variable; and α is the intercept. The coefficients β correspond to the focal variables $z_i = \{z_{mi}\}_{m=1}^{M}$ (coded from unstructured data), while γ corresponds to the observed covariates $x_i = \{x_{li}\}_{l=1}^{L}$ (arising from structured data). The structural error term ε_i is assumed to have zero mean and conditional variance $\mathrm{Var}(\varepsilon_i \mid x_i, z_i) = \sigma^2_{\varepsilon,i}$, which may vary across observations (accommodating heteroskedasticity). The complete parameter vector

is $\delta = \{\alpha, \beta, \gamma\}$. For brevity, we suppress the observation index i. Finally, our analysis applies to both scalar-valued y (in single-equation models) and vector-valued y (in system-of-equations models); we use lowercase letters for scalars and uppercase letters for vectors or matrices.

In between-subjects experimental studies, where participants are shown different stimuli based on their experimental condition—so that only one variable (e.g., humor) varies across conditions—we have $M = 1; L = 0$, and the analysis of variance (ANOVA) reduces to testing whether $\beta_1 \neq 0$. When additional observed covariates are present (e.g., when some $x_{li} \neq 0$), analysis of covariance (ANCOVA) similarly involves testing the hypothesis that $\beta_1 \neq 0$.

The structured variables x are available for the entire sample. Because manually coding z (and potentially w) for the entire dataset is impractical, we assume a calibration subsample has been manually-coded in Step 2, while the entire dataset has been coded by AI in Step 3. Consequently, the manually-coded values of z are available only in the calibration subsample, whereas $\hat{z}$, its AI-coded counterpart, is available for the full dataset, including the calibration subsample. Since AI systems operate as statistical models, their application introduces coding errors—discrepancies between z (from Step 2) and $\hat{z}$ (from Step 3). The calibration subsample allows us to model these coding errors and ensure consistent inference in the full dataset.

We specify the following observation model, which explicitly defines the relationship between the AI-coded variables ($\hat{z}$ and $\hat{w}$), the true but unobserved values of those variables (z and w), and the AI-coding errors (η_z and η_w):

$$\hat{z}_m = z_m + \eta_{zm}; \quad \hat{w}_p = w_p + \eta_{wp}. \tag{2}$$

where $\hat{z} = \{\hat{z}_m\}_{m=1}^{M}$ and $\hat{w} = \{\hat{w}_p\}_{p=1}^{P}$ are the AI-coded variants of the conceptual variables z and instruments w. The terms $\eta_z = \{\eta_{zm}\}$ and $\eta_w = \{\eta_{wp}\}$ represent the AI-coding errors.

Unlike traditional models—which often assume that coding errors (η_z and η_w) are independent of the regressors (x and z) and any instruments (w) (i.e., $\eta_z, \eta_w \perp\!\!\!\perp x, z, w$)—our model allows for correlations between the coding errors and the regressors, as well as the instruments. While

these classical assumptions are often reasonable in well-designed survey or experimental contexts (Abel 2017; Carroll et al. 2006), the accuracy of AI-coded variables often varies with key data characteristics—such as resolution, background noise, or even the race of individuals in an image. For example, AI models have been shown to exhibit higher misclassification rates when processing low-resolution images, noisy audio tracks, or photos of people of color (Buolamwini and Gebru 2018). When such characteristics are also included as explanatory variables in an econometric model, the resulting correlations between AI-coding errors and the regressors can introduce complexities that traditional models fail to fully address. These issues can lead to *both* Type I (false positive) and Type II (false negative) errors in hypothesis testing, resulting in erroneous theory development where spurious causal relationships are inferred and true causal relationships are overlooked.

While some recent approaches relax the classical assumption of error independence—allowing errors to depend on observable or latent traits (Gustafson 2003)—much of this literature assumes that the systematic components driving bias are latent. These components must then be corrected indirectly through additional identifying assumptions, external instruments, or methods such as simulation extrapolation (e.g., Cook and Stefanski 1994; Schennach 2004). For example, Bound, Brown, and Mathiowetz (2001) account for survey respondents' misreporting of wages by modeling their true wage levels, Hu (2008) show that nonclassical errors can be addressed under certain conditions if multiple noisy measures of the same (latent) construct are available, and Cunha, Heckman, and Schennach (2010) describe a dynamic factor model of skill formation in the presence of measurement error in proxies for latent variables such as skills and parental investments. Other researchers, such as Sepanski and Carroll (1993) and Chen, Hong, and Tamer (2005), propose semi-parametric methods like sieve estimation. While theoretically rich, these methods require prohibitively large samples in practice due to their relatively slower rates of convergence (see Schennach 2016 for a recent review).

In contrast, a key insight of our approach is that AI-coded applications provide researchers with a unique advantage: access to calibration subsamples that reveal systematic patterns in

coding errors. Unlike traditional models, which typically treat the sources of systematic bias as unobservable, AI-coded data allow researchers to directly model the relationship between AI-coding errors and observable data features—such as image resolution, lighting conditions, or AI model-specific biases.

This provides two key benefits. First, it enables the estimation of the systematic component of the error, facilitating *direct bias correction* with fewer restrictive assumptions. Second, it uniquely allows for coding errors in instruments and correlations between instruments and coding errors, thereby accomodating the use of *AI-coded instruments* (which are likely to coded with error for the same reasons as the regressors). In contrast, most existing methods require instruments to be coded without error, precluding the use of AI-coded instruments.

These distinctions underscore the novel opportunities afforded by AI-coded data, as leveraged in our modeling framework, to address nonclassical measurement error in ways that are infeasible in many traditional settings. In the next subsection, we present the specifics of two estimators—one for a single-equation model and the other for a system-of-equations model—along with associated algorithms for inference.

3.2.1 Bias-Corrected Estimators: Derivation and Development

To address the challenges posed by AI-coding errors, we develop estimators that explicitly account for potential correlations between these errors and the regressors and instruments. Our approach involves adjusting canonical estimators by incorporating estimates of key moments—specifically, cross-product matrices—of the coding errors and the relevant variables.

Specifically, in Web Appendix §C, we focus on single-equation models, which involve a single dependent variable. We begin by deriving the bias that arises when coding errors are ignored. We then adjust the canonical two-stage least squares (2SLS) estimator—which encompasses ordinary least squares (OLS), weighted least squares (WLS), and generalized least squares (GLS) as special cases—to obtain consistent estimates. We prove consistency and asymptotic normality, and we derive the necessary identification conditions for our method, including requirements such as

the full rank of certain matrices and the informativeness of instruments. Finally, we present an algorithm (Algorithm 6) that outlines the steps to estimate the bias-corrected parameters and compute standard errors. In the algorithm, nB is the number of bootstrap iterations, Data_s denotes the manually coded calibration subsample, and Data_f denotes the full sample. N_s and N_f are the number of bootstrap draws from Data_s and Data_f, respectively.

Algorithm 1 Algorithm for Inference in Single Equation Marketing Models

Input: Manually coded calibration subsample (Data_s), AI-coded full sample (Data_f).

▷ Steps 1-3 (data acquisition, manual coding, AI coding) are assumed to be complete.

for $b \leftarrow 1, nB$ **do**

Draw Bootstrap Calibration Subsample$[b]$ of size N_s from Data_s with replacement.

Infer $\hat{\psi}_{\text{2SLS}} = \frac{\hat{\zeta}'\eta_\xi}{N_s}$ in Bootstrap Calibration Subsample$[b]$.

Draw Bootstrap Full Sample$[b]$ of size N_f from Data_f with replacement.

Compute the canonical 2SLS estimator $d_{\text{2SLS}}[b]$ in Bootstrap Full Sample$[b]$.

Infer $\hat{s}_{\xi\zeta} = \frac{\hat{\xi}'\hat{\zeta}}{N_f}$ and $\hat{s}_{\zeta\zeta} = \frac{\hat{\zeta}'\hat{\zeta}}{N_f}$ in Bootstrap Full Sample$[b]$.

Calculate $\widehat{\text{BiasFactor}}_{\text{2SLS}}[b] = \left[\hat{s}_{\xi\zeta}\,\hat{s}_{\zeta\zeta}^{-1}\,\hat{s}_{\xi\zeta}'\right]^{-1}\hat{s}_{\xi\zeta}\,\hat{s}_{\zeta\zeta}^{-1}\,\hat{\psi}_{\text{2SLS}}$.

$d_{\text{2SLS}}^{con}[b] = (I_g - \widehat{\text{BiasFactor}}_{\text{2SLS}}[b])^{-1} d_{\text{2SLS}}[b]$.

end for

Infer test statistics from the distribution of $d_{\text{2SLS}}^{con}[b], \quad b = 1, \ldots, nB$.

In Algorithm 6, the matrix $\xi = [z\ x]$ is the $N \times g$ *true* design matrix, which includes both the true values of the AI-coded regressors (corresponding to the z_{mi} variables in Equation (1)) and the observed regressors (corresponding to the x_{li} variables in Equation (1)). The $N \times h$ matrix ζ is the *true* instrument matrix, with $h \geq g$, and includes *all* instruments: (1) any exogenous regressors from x that are also valid instruments, (2) the *true* values of any exogenous regressors from z that are also valid instruments, and (3) the *true* values of any variables, w, that are used *only* as instruments (and are not included in x or z). $\hat{\xi} = [\hat{z}\ x]$ is the observed design matrix, and $\hat{\zeta}$ is the observed instrument matrix.

Web Appendix §D extends our framework to system-of-equations marketing models, such as those estimated using three-stage least squares (3SLS). These models are particularly relevant when multiple interrelated outcomes share common drivers or when structural errors across

equations are correlated—for example, in studies that jointly estimate demand and supply. AI-coding errors in such settings pose additional challenges because they can propagate across equations, compounding biases in parameter estimates.

Building on our single-equation results, we adapt Algorithm 6 to the system-of-equations context, developing consistent estimators that adjust for both within-equation and cross-equation biases. As with the single-equation case, we prove consistency, asymptotic normality, and derive the necessary identification conditions. We present the method in Algorithm 2. For brevity, we define equation objects in the algorithm, including $A_{3\text{SLS}}^{-1}$ and $B_{3\text{SLS}}$, in the Web Appendix. Data_s denotes the manually coded subsample, and Data_f denotes the full sample. N_s and N_f represent the number of bootstrap draws from Data_s and Data_f, respectively.

Algorithm 2 Algorithm for Inference in System-of-Equations Marketing Models

Input: Manually coded calibration subsample (Data_s), AI-coded full sample (Data_f).

▷ Steps 1-3 (data acquisition, manual coding, AI coding) are assumed to be complete.

for $b \leftarrow 1, nB$ **do**

Draw Bootstrap Subsample$[b]$ of size N_s from Data_s.

Infer $\hat{\psi}_g = \frac{\vec{\hat{\zeta}}' \eta_{\xi g}}{N_s}, g = 1, \ldots G$ in Bootstrap Subsample$[b]$.

Draw Bootstrap Full Sample$[b]$ of size N_f from Data_f.

Compute the canonical 3SLS estimator $D_{3\text{SLS}}[b]$ in Bootstrap Full Sample$[b]$.

Infer $\hat{s}_{\xi_i \zeta}$ for $i = 1, \ldots, G$ and $\hat{s}_{\zeta\zeta}$ in Bootstrap Full Sample$[b]$.

Infer the estimation analogs of $A_{3\text{SLS}}$ and $B_{3\text{SLS}}$: $\hat{A}_{3\text{SLS}}$ and $\hat{B}_{3\text{SLS}}$.

Compute $\widehat{\text{BiasFactor}}_{3\text{SLS}} = \hat{A}_{3\text{SLS}}^{-1} \hat{B}_{3\text{SLS}}$.

Solve $\left[I_{Gg} - \widehat{\text{BiasFactor}}_{3\text{SLS}} \right] D_{3\text{SLS}}^{\text{con}}[b] = D_{3\text{SLS}}[b]$ in Bootstrap Full Sample$[b]$.

end for

Infer test statistics from the distribution of $D_{3\text{SLS}}^{\text{con}}[b], \quad b = 1, \ldots, nB$.

Although the derivation and proof of these estimators involve substantial linear algebra, and the model setup for Algorithm 2 is notationally cumbersome (thus presented only in the Web Appendix), the final estimators are computationally efficient and straightforward to implement. In particular, we show in the appendix that canonical estimators (2SLS and 3SLS) project the true parameters onto the space spanned by the instruments. Our proposed correction uses standard linear algebra to project these estimates back, ensuring consistency. Because these operations can

be performed using widely available statistical software (e.g., numpy in Python and base matrix libraries in R), our inference strategy is both scalable and highly accessible, as illustrated in the implementation guide (Web Appendix §B).

3.2.2 Doubly Robust Estimation: Correcting for Manual-Coding Errors

Manually coded variables, often treated as validation benchmarks, can themselves exhibit imprecision—particularly for subjective constructs like humor, where individual coders may interpret and rate the same stimulus differently. This variability introduces noise into the manually coded measures, and if unaddressed, risks biasing even the bias-corrected estimators proposed in this study.

To address this dual challenge of AI- and manual-coding errors, in Web Appendix §E, we integrate a Simulation–Extrapolation (SIMEX) procedure (Cook and Stefanski 1994) into our framework (described in Algorithm 3). This approach models and adjusts for imprecision in manual coding in two stages:

1. **Simulation Step:** We systematically introduce increasing levels of synthetic noise into the manually coded variables, generating datasets with progressively higher measurement error. For each noisy dataset, we re-estimate the model parameters using our bias-corrected estimator, tracking how the estimates (both means and critical values) evolve as imprecision increases.

2. **Extrapolation Step:** Using the trajectory of parameter estimates from the simulation step, we extrapolate backward to infer the estimates that would arise in the absence of manual-coding error. This back-extrapolation effectively "removes" the imprecision, yielding corrected parameter (and critical) values that are robust to variability in human judgment.

Algorithm 3 builds upon the original SIMEX method of Cook and Stefanski (1994) and its subsequent adaptations (e.g., Yang et al. 2018), but extends it in a crucial way: it simultaneously corrects for *both* AI-coding error *and* imprecision in the manually coded calibration data. Prior

Algorithm 3 Doubly Robust Algorithm: Addressing Imprecision in the Manually Coded Measures

Input: Manually coded calibration subsample (Data_s), AI-coded full sample (Data_f).

▷ Steps 1-3 (data acquisition, manual coding, AI coding) are assumed to be complete.

for $k \leftarrow 1, K$ **do**

for $b \leftarrow 1, nB$ **do**

Draw a bootstrap sample Bootstrap Subsample$[b]$ of size N_s from Data_s.

Simulate imprecision corresponding to k.

Add simulated imprecision to the manually coded measures in Bootstrap Subsample$[b]$.

Infer limit matrices and densities in Bootstrap Subsample$[b]$.

Draw a bootstrap sample Bootstrap Full Sample$[b]$ of size N_f from Data_f.

Compute $d_{\text{2SLS}}[b,k]$ or $D_{\text{3SLS}}[b,k]$ using limit matrices in Bootstrap Full Sample$[b]$.

end for

Compute estimates and critical values of $d_{\text{2SLS}}[,k]$ or $D_{\text{3SLS}}[,k]$ as d_k or D_k.

end for

Model the estimates and critical values, $d_k, k = 1, \ldots K$, as a function of k and extrapolate to the case of no imprecision.

SIMEX estimators were designed to handle only a single source of measurement error, typically assuming that any calibration or validation data are error-free. This means they could be applied to *either* AI-coding errors *or* manual-coding errors, but not to both simultaneously. Our method, in contrast, is explicitly designed to address both sources of error within a unified framework.

A key limitation of SIMEX applications is their reliance on the assumption that measurement error follows an additive, scaled parametric distribution—often a Gaussian distribution, as in Cook and Stefanski (1994), or a linear transformation of a Gaussian, as in Yang et al. (2018). These assumptions are well-suited to manually coded measures, where independent ratings from multiple human coders can be averaged, and the Central Limit Theorem suggests that such averages will tend towards a normal distribution. However, as previously discussed, AI-coded variables often deviate from true values in more complex, systematic ways that do not conform to these distributional assumptions.

Therefore, while SIMEX is a natural choice for addressing human-coding error, it is less directly applicable to the AI-coding error itself. Our method explicitly recognizes this distinction.

We impose *fewer* restrictions on the AI-coding errors, requiring only that their first and second moments exist and are well-defined, rather than assuming a specific parametric distribution. We then apply the SIMEX assumptions to the *manually coded* measures in the calibration subsample, where they are more appropriate.

This distinction is critical for the validity and robustness of our approach. By coupling the SIMEX correction for manual coding imprecision with our bias-corrected estimators for AI-coding error (developed in Web Appendix §C and D), we achieve a *doubly robust* estimation procedure. It is scalable and precise because it leverages AI coding for the bulk of the data, yet it remains reliable in its inference because it uses SIMEX-adjusted manual coding to correct for both sources of error. This allows researchers to integrate large-scale, AI-coded data into econometric models, even when both the AI coding and the manual coding used for calibration are imperfect.

4 Monte Carlo Evidence

To evaluate our proposed estimators, we conduct simulation studies that closely replicate consumer research examining the impact of nostalgia appeals on consumer engagement and brand attitude. In consumer research, nostalgia—a sentimental longing for an idealized past—is a powerful emotional construct that shapes how individuals perceive and interact with brands (Holbrook and Schindler 1991). By activating cherished personal or collective memories, nostalgia can forge stronger affective bonds between consumers and products or brands that successfully evoke these reflections.

Nostalgia appeals are particularly relevant to consumer engagement and brand attitude—two constructs central to consumer psychology (e.g., Muehling, Sprott, and Sultan 2014). Consumer engagement metrics (e.g., click-through rates, dwell times, social media interactions) gauge how actively consumers interact with brand communications, while brand attitude reflects longer-term evaluative judgments toward the brand. Both constructs can be heightened by positive emotions, especially those evoking personal or collective memories (Hirsch 1992).

Nostalgic appeals also influence consumers' self-concept and identity; for instance, people with a stronger need to belong exhibit greater preferences for nostalgic products (Loveland, Smeesters, and Mandel 2010). By conjuring warm, comforting feelings that foster trust and loyalty, nostalgia can bolster brand attitude (Holak and Havlena 1998). Additionally, Lasaleta, Sedikides, and Vohs (2014) show that activating nostalgic memories reduces attachment to money, increasing willingness to spend on products that evoke such sentiments. Collectively, these findings suggest that nostalgia serves not only as an emotional driver of consumer engagement but also as a powerful mechanism shaping identity and consumer–brand relationships.

4.1 Single-Equation Model

We model consumer engagement in the single-equation model as:

$$\text{Engagement} = \beta_{\text{nost}} z_{\text{nost}} + \gamma_{\text{fea}} x_{\text{fea}} + \varepsilon. \tag{3}$$

Here, Engagement is an N-vector of dependent variable observations; ε is an unobservable, mean-zero N-vector of independently and identically distributed random variables with variance σ_ε^2; and N is the sample size. The coefficient vector is $\delta = \{\beta_{\text{nost}}, \gamma_{\text{fea}}\}$, and the design matrix of regressors is $\xi = \{x_{\text{fea}}, z_{\text{nost}}\}$. We use mean-centered regressors and set all coefficients to 1 to simplify the exposition; these considerations do not affect the generality of our results. When studying system-of-equations models, we introduce analogous equations.

The variables in this model, and the instruments used in later studies, are defined as follows:

1. z_{nost} is the level of nostalgia appeal, serving as a key *independent variable* that is unobserved and must be coded from unstructured data. For example, nostalgia appeals may be coded using AI systems from video advertisements.
2. x_{fea} is the value of an objective product feature, acting as a *control variable* that is observed and measured without error.
3. w_{her} is an AI-coded proxy variable for the availability of historical assets that serves as an

instrumental variable. It summarizes the availability of audio-visual material relating to the brand's history, founding stories (e.g., the brand's origin story), and vintage products, which can be used to create nostalgia appeal.

4. w_{cost} is another AI-coded proxy variable for the cost of producing nostalgic content, derived from ad visuals, text, and audio complexity. It also serves as an instrumental variable, reflecting the financial or technical effort required to create retro-themed advertisements, such as licensing old footage or designing vintage packaging.

4.2 Simulation Scenarios

Our approach is motivated by the observation that, in naturalistic settings, nostalgia is an example of an emotional appeal that is not directly observable but must be inferred from rich, unstructured data like images, text, and video. Therefore, measuring nostalgia from real-world unstructured data can benefit from AI-based coding, which may introduce coding error.

We conduct eight distinct studies—four with single-equation models and four with system-of-equations models—to examine the influence of coding error and the efficacy of our proposed corrections. To simplify the exposition, we assume all regressions share the same independent variables and instruments in our simulations, although our overall framework allows different equations to include different sets of regressors or instruments. We consider the following four scenarios for each model:

Study 1: All regressors are exogenous and coded without error; we expect all estimators to be consistent.

Study 2: We introduce coding imprecision in z_{nost} that is correlated with x_{fea}. This study relates to cases where the accuracy of nostalgia coding varies with the features of the product, such that the nostalgia intensity of some products can be determined more precisely than others. As the independent variables are correlated, we expect the canonical estimators to be biased for both coefficients β_{nost} and γ_{fea}, even though only z_{nost} is coded with error.

Study 3: We introduce endogeneity in z_{nost} and include instruments (w_{her} and w_{cost}). We expect

the regression estimator to be inconsistent. As the instruments are coded without error, we expect the 2SLS estimator to be consistent for β_{nost} but inconsistent for γ_{fea}, due to the correlation between z_{nost} and x_{fea}.

Study 4: We introduce coding imprecision in the instruments (w_{her} and w_{cost}). This scenario aligns with the expectation that the accuracy of AI coding of social media and advertising content may vary with product features (e.g., the product's brand), such that the AI may be more or less precise in coding heritage mentions, production costs of some brands, or the cost of producing nostalgic content.

4.3 Implementation

For each study, we construct a dataset of 50,000 observations based on the same random number seed. The first 2,500 observations (5% of the data) represent the calibration subsample where z_{nost} is manually coded; all AI-coded measures are available in the complete sample. Our initial studies assume the manual coding is precise, meaning that there is no coding error in the manually-coded data. In subsequent studies, we revisit this assumption, introducing coding imprecision in the manually-coded measures and investigating its impact on our estimator's performance. Studies 2, 3, and 4 explore AI coding imprecision. Studies 1 and 2 impose exogeneity, while Studies 3 and 4 consider endogeneity. Across all studies, we expect our proposed estimator to be consistent, as it is designed to account for various complex forms of coding error in situations with and without endogenous variables.

We impose these restrictions as follows. We consider each variable as a composition of three components:

1. The **exogenous component** of each variable that is orthogonal to the structural error term and potentially correlated with the coding error component of other variables.

2. The **endogenous component** of each variable that is non-zero and correlated with the structural error term for endogenous variables, and zero for exogenous variables and instru-

ments.

3. The **coding error component** of each variable that is zero for variables coded without error, and non-zero and potentially correlated with other variables for variables coded with error.

We generate data on each of these components from a mean-zero multivariate normal distribution, adjusting the covariance matrix to match each study's specific conditions. The covariance matrix governs the relationships between variable components, including endogeneity and coding error correlations. When a component should be zero (such as the endogenous component in exogenous variables), we set its corresponding covariance terms to zero, yielding a zero vector for that component. For example, in Study 2, we induce a correlation between the exogenous component of x_{fea} and the coding error in z_{nost}, while setting both the endogenous and coding error components of x_{fea} to zero, as x_{fea} is an exogenous variable measured without error. Similarly, in Study 3, we induce a correlation between the endogenous component of z_{nost} and ε to reflect endogeneity. This approach enables us to simulate data that precisely matches the conditions specified in each study.

To facilitate replication, we provide our simulation code in Web Appendix §F. The programs are self-contained and thoroughly documented; users will need to install the required libraries and modify file paths for their local environment. They are provided in R because they serve two purposes: detailing the simulations and providing a practical implementation of our framework. To this end, they are complementary to our implementation guide, which uses Python code, in facilitating the adoption of these methods.

4.4 Single-Equation Model Results

In Study 1, where all regressors are exogenous and coded with precision, all estimators—including the regression estimator, the 2SLS estimator, and our proposed estimator—are consistent. The estimates for both β_{nost} and γ_{fea} are close to the true value of 1, as shown in Table 2, and both fall

Table 2: Single-Equation Model

		Studies			
		(1)	(2)	(3)	(4)
β_{nost}	Regression	1.003	0.541	0.774	0.774
		(0.004)	(0.004)	(0.003)	(0.003)
	2SLS	1.003	0.541	1.004	1.259
		(0.004)	(0.004)	(0.008)	(0.016)
	Proposed	1.004	0.952	1.027	1.07
		(0.005)	(0.029)	(0.027)	(0.043)
γ_{fea}	Regression	1.006	0.788	0.695	0.694
		(0.004)	(0.006)	(0.006)	(0.006)
	2SLS	1.006	0.788	0.603	0.500
		(0.004)	(0.006)	(0.007)	(0.009)
	Proposed	1.006	0.986	0.984	0.993
		(0.005)	(0.021)	(0.023)	(0.024)

Estimates for β_{nost} and γ_{fea} across four studies. The true coefficient is 1. Standard errors are in parentheses. Estimates in red indicate that the true value is outside the 99% confidence interval.

within the confidence intervals.

In Study 2, z_{nost} (nostalgia) is exogenous but coded with imprecision, and this imprecision is correlated with x_{fea} (product feature). In this case, both the regression estimator and the 2SLS estimator are inconsistent, yielding biased estimates for β_{nost} and γ_{fea}. This demonstrates how AI-coding imprecision can affect multiple coefficients when the imprecision is correlated with other regressors. Specifically, the imprecision in z_{nost} biases the estimate of β_{nost}. Moreover, because the imprecision is correlated with x_{fea}, it also biases the estimate of γ_{fea}. The 2SLS estimator fails to correct these issues, as it lacks the appropriate instruments.

In Study 3, z_{nost} is also endogenous, meaning it correlates with the error term in the regression equation. Here, we introduce instrumental variables (w_{her} and w_{cost}) to address both endogeneity and coding imprecision. The 2SLS estimator succeeds in producing a consistent estimate for β_{nost} (with a value of 1.004), effectively correcting for the endogeneity of nostalgia intensity. However, the estimate for γ_{fea} remains biased (with a value of 0.603), illustrating that coding imprecision in one variable can contaminate the inference of other coefficients, even if the endogeneity of the

variable coded with error is corrected.

In Study 4, we introduce coding imprecision in the instruments, replicating a scenario where the instruments are constructed using AI and are therefore subject to coding error. In this case, the 2SLS estimator is inconsistent for both β_{nost} and γ_{fea}, as the coding errors in the instruments lead to violations of the conditions for valid instruments. Specifically, the coding errors in the instruments are correlated with the included regressors, leading to the observed inconsistency.

Our proposed estimator remains consistent across all four studies. In each case, it yields coefficient estimates for β_{nost} and γ_{fea} that are close to the true value of 1 and within the confidence intervals. Thus, it effectively accounts for coding imprecision in both the independent variables and instruments, mitigating the biases that affect the regression and 2SLS estimators.

4.5 System-of-Equations Model Results

System-of-equations models are well-suited to capturing real-world marketing scenarios in which multiple related outcomes are influenced by common factors. To explore the impact of nostalgia appeals on both *consumer engagement* (as defined in the single-equation model) and overall *brand attitude*, we extend the previous model by adding an equation for Brand Attitude:

$$\text{Engagement} = z_{\text{nost}}\beta_{\text{eng, nost}} + x_{\text{fea}}\gamma_{\text{eng, fea}} + \varepsilon_{\text{eng}}, \tag{4}$$

$$\text{Brand Attitude} = z_{\text{nost}}\beta_{\text{brand, nost}} + x_{\text{fea}}\gamma_{\text{brand, fea}} + \varepsilon_{\text{brand}}. \tag{5}$$

In these equations, Engagement, z_nost, and x_fea retain their definitions from the single-equation model. Brand Attitude represents the consumer's overall attitude toward the advertised brand. The coefficients $\beta_{\text{eng, nost}}$ and $\gamma_{\text{eng, fea}}$ are the effects of nostalgia and product features, respectively, on engagement, while $\beta_{\text{brand, nost}}$ and $\gamma_{\text{brand, fea}}$ represent the analogous effects on brand attitude. ε_{eng} and $\varepsilon_{\text{brand}}$ are the structural error terms for the Engagement and Brand Attitude equations, respectively.

Unobserved factors, such as an individual's inherent preference for nostalgic brands or products, might simultaneously influence both their engagement with a nostalgic ad and their overall attitude towards the brand. To capture this dependency between these outcomes, we introduce a correlation of 0.25 between the structural errors, ε_{eng} and $\varepsilon_{\text{brand}}$.

We conduct four simulation studies that parallel those in the single-equation model, assessing the performance of the canonical 3SLS estimator and our proposed estimator under varying conditions of AI-coding errors and endogeneity, using the same instruments, w_{her} and w_{cost}, to address potential endogeneity. Analogous to our single-equation model simulations, we generate data on the variable components from a mean-zero multivariate Gaussian distribution, adjusting the covariance matrix to match the specific conditions of each study. Table 3 presents our results.

Table 3: System-of-Equations Model

		Studies			
		(1)	(2)	(3)	(4)
$\beta_{\text{eng, nost}}$	3SLS	1.002	0.543	1.003	1.262
		(0.004)	(0.004)	(0.008)	(0.015)
	Proposed	1.002	1.007	1.026	1.02
		(0.005)	(0.035)	(0.029)	(0.04)
$\gamma_{\text{eng, fea}}$	3SLS	1.002	0.783	0.597	0.494
		(0.004)	(0.006)	(0.007)	(0.009)
	Proposed	1.001	0.998	0.998	0.996
		(0.005)	(0.023)	(0.023)	(0.023)
$\beta_{\text{brand, nost}}$	3SLS	0.999	0.54	1	1.251
		(0.004)	(0.004)	(0.008)	(0.015)
	Proposed	0.999	1.005	1.022	1.011
		(0.004)	(0.035)	(0.028)	(0.039)
$\gamma_{\text{brand, fea}}$	3SLS	0.996	0.778	0.592	0.491
		(0.004)	(0.006)	(0.007)	(0.009)
	Proposed	0.996	0.993	0.992	0.989
		(0.005)	(0.023)	(0.023)	(0.023)

Estimates for $\beta_{\text{eng, nost}}$, $\gamma_{\text{eng, fea}}$, $\beta_{\text{brand, nost}}$, and $\gamma_{\text{brand, fea}}$ across four studies. The true coefficient is 1. Standard errors are in parentheses. Estimates in red indicate that the true value is outside the 99% confidence interval.

Study 1 serves as our baseline, with all variables exogenous and coded without error. The

estimates for $\beta_{\text{eng, nost}}$, $\gamma_{\text{eng, fea}}$, $\beta_{\text{brand, nost}}$, and $\gamma_{\text{brand, fea}}$ are close to the true value of 1, with small standard errors. Thus, under canonical conditions, the 3SLS estimator performs well, yielding accurate estimates for all coefficients, consistent with theoretical expectations.

Study 2 introduces AI-coding imprecision in z_{nost}, which is correlated with x_{fea}. The 3SLS estimator becomes biased due to this coding error. The imprecision in z_{nost} propagates through *both* the Engagement and Brand Attitude equations because of the shared variables and the error correlation between equations. Consequently, the estimates of all four coefficients—$\beta_{\text{eng, nost}}$, $\gamma_{\text{eng, fea}}$, $\beta_{\text{brand, nost}}$, and $\gamma_{\text{brand, fea}}$—are biased. Our proposed estimator effectively adjusts for this bias, producing estimates that are statistically indistinguishable from the true values.

Study 3 introduces endogeneity in z_{nost} and includes instruments coded without error (w_{her} and w_{cost}). The 3SLS estimator adequately addresses the endogeneity for z_{nost}, but the estimates for $\gamma_{\text{eng, fea}}$ and $\gamma_{\text{brand, fea}}$ remain biased due to the uncorrected correlation between the coding errors and x_{fea}. Our proposed estimator accounts for both endogeneity and coding imprecision, yielding consistent estimates for all coefficients.

Study 4 simulates coding imprecision in the instruments, which is correlated with x_{fea}. This scenario leads to overestimated coefficients for z_{nost} and underestimated coefficients for x_{fea}. The 3SLS estimator fails to provide reliable estimates because the validity of the instruments is compromised by the coding errors. Our proposed estimator corrects for these issues, providing estimates that are much closer to the true values and within the confidence intervals.

Overall, the results from the system-of-equations model parallel those from the single-equation model, highlighting that traditional estimators are vulnerable to biases when AI-coded variables are used as independent variables or instruments without accounting for coding errors. Our proposed estimator consistently delivers accurate and reliable estimates across all scenarios, demonstrating its robustness and practical utility in complex models prevalent in marketing research.

4.6 Implications of Imprecision in the Manually-Coded Measures

Figure 3 illustrates the effects of increasing imprecision in the manually-coded measures in Study 4, which represents a scenario encompassing the full range of potential issues with AI-coding in marketing research: endogeneity, coding errors in the independent variables, and coding errors in the instruments. We generate results for a grid of 201 data points, with imprecision increasing in 0.01 increments from 0 to 2. The black points represent the estimates, the blue line shows the fit from a smooth regression, and the gray bands indicate the associated standard errors. To ensure comparability, we focus on estimates of the coefficient on nostalgia intensity—β_{nost} in the single-equation model and $\beta_{\text{eng, nost}}$ in the system-of-equations model. The true value of both coefficients is 1.

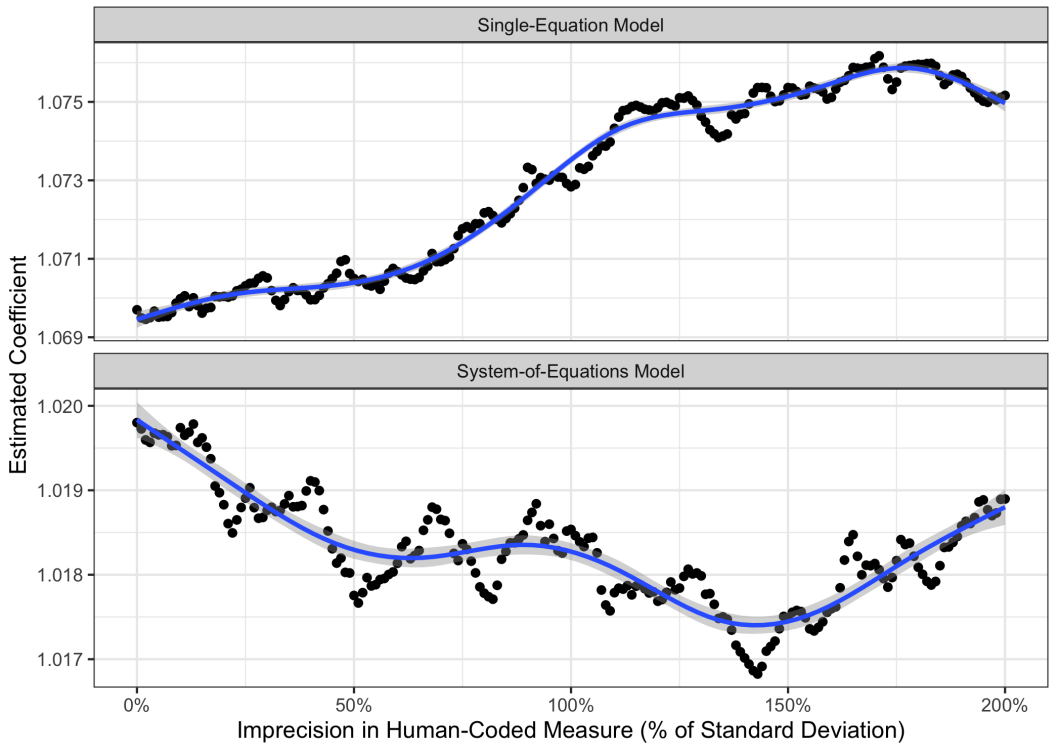


Figure 3: Implications of Imprecision in the Manually-Coded Measures on Estimates

Black points represent the estimates; the blue line shows the GAM fit, and the gray bands indicate standard errors. The x-axis represents imprecision in the manually-coded measure as a percentage of the standard deviation of the conceptual variable.

As expected, the performance of the estimators deteriorates as imprecision increases; however,

the substantive impact remains limited across a wide range of conditions. This robustness is likely because our proposed estimators require the inference of only a few cross-product terms for bias correction, making them relatively resilient to the presence of imprecision in the manually-coded measures. In contrast, the canonical estimator diverges in both the single-equation and system-of-equations models (see Table 2 and Table 3). Moreover, for the canonical estimators, the inclusion or exclusion of manually-coded measures in a subsample of observations has little effect on performance, as the imprecision of the AI-coded samples overwhelms the more precise manual coding.

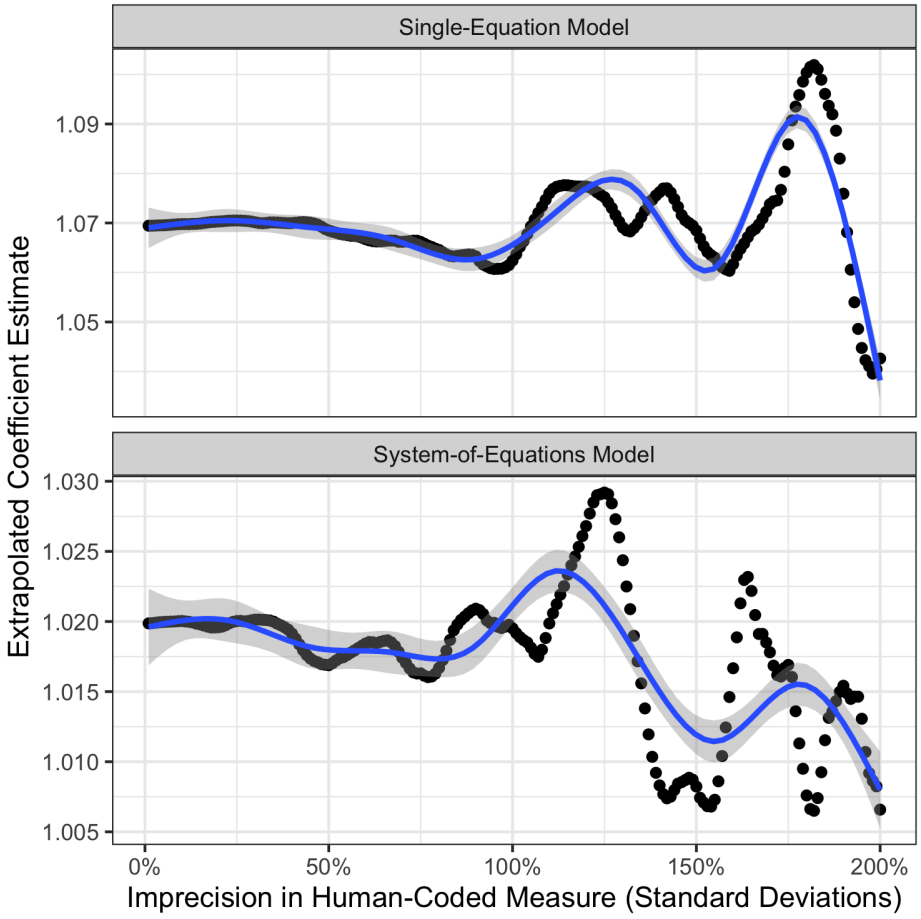


Figure 4: Correcting for Imprecision in the Manually-Coded Measures

Black points represent the extrapolated estimates; the blue line shows the GAM fit, and the gray bands indicate standard errors. True coefficient value = 1. The x-axis represents imprecision in the manually-coded measure as a percentage of the standard deviation of the conceptual variable.

To evaluate our back-extrapolation procedure, we analyze datasets with varying levels of

coding imprecision, indexed by $i^* \geq 2$, where higher values of i^* correspond to greater imprecision in the manually-coded measures. For each i^*, we generate additional synthetic datasets with progressively higher imprecision and apply our proposed estimators. We then fit a Generalized Additive Model (GAM) to the estimates from these datasets, enabling us to back-extrapolate to the case of no imprecision (i.e., $i^* = 0$).[1]

Figure 4 demonstrates that our imprecision correction procedure is highly effective, yielding results that align closely with the true coefficients across a wide range of manual-coding imprecision levels. Specifically, for datasets where the signal-to-noise ratio is less than or equal to 1—that is, the standard deviation of coding error (noise) is less than or equal to the standard deviation of the true construct (signal)—both our proposed estimators and the extrapolation procedure provide effective remedies. At lower levels of noise, the extrapolated estimates are closer to the true values. As i^* increases, the performance of the back-extrapolation deteriorates: higher levels of noise necessitate more extensive extrapolation, making precise estimation more challenging. When the signal-to-noise ratio exceeds 1, the extrapolated estimates remain substantively meaningful—they stay close to the true value of 1—but become increasingly dispersed. In such cases, the raw estimates from our proposed estimators are more effective as the terms inferred from the manually-coded data are linear. Therefore, any noise is averaged out effectively in the raw estimates.

5 Study: Physical Appeal as a Driver of Social Media Engagement

Physical appeal, characterized by attractiveness and aesthetic presentation, has long been recognized as a significant factor in consumer behavior and marketing. Seminal work by Baker and Churchill Jr (1977) demonstrated that the physical attractiveness of models in print advertisements

[1] While we chose a GAM for its flexibility in modeling non-linear relationships without strong parametric assumptions, the low variance of the estimates around the true values (as seen in the figures) suggests that simpler models (e.g., linear or cubic spline regression) would likely yield similar conclusions.

positively influences consumers' evaluations of both the advertisements and the advertised products. More recently, Aagerup (2011) found that the use of idealized beauty in fashion advertising affects consumer self-esteem and purchase intentions. Furthermore, brand concepts that effectively leverage physical appeal can serve as representations of human values, where cultural congruity enhances perceived authenticity and, consequently, consumer engagement (Torelli et al. 2012).

The rise of social media platforms has transformed brand communication and consumer engagement. Users are no longer passive recipients of curated brand messages but active participants in content creation and dissemination—a shift that Ritzer, Dean, and Jurgenson (2012) termed "prosumption," where the boundaries between producers and consumers blur. As users curate their digital identities—engaging in an extension of self in the digital world (Belk 2013)—physical appearance plays a crucial role. Visual social media content, therefore, tends to elicit stronger emotional responses and higher engagement than text-based content (Pentina, Guilloux, and Micu 2018). Supporting this, Bakhshi, Shamma, and Gilbert (2014) found that Instagram posts featuring faces are 38% more likely to receive likes and 32% more likely to receive comments, regardless of the subject's age or gender.

In this study, we investigate the impact of physical appeal on user engagement on Instagram—a leading social media platform. Although theory suggests that physical appeal plays a significant role in driving engagement, its influence in real-world settings—characterized by diverse content and complex user interactions—remains understudied. Moreover, on visually driven platforms such as Instagram, physical appeal can also operate as a form of nonverbal communication, thereby further amplifying its influence.

Our research builds on recent work examining the dynamics of social media engagement. De Vries, Gensler, and Leeflang (2012) investigated the drivers of brand post popularity on social networking sites, finding that the vividness and interactivity of posts significantly influence engagement. Berger and Milkman (2012) explored how content characteristics affect the virality of online content, noting that content evoking high-arousal positive emotions is more likely to go viral. Extending these lines of inquiry—and highlighting the role of perceived authenticity—

Valsesia and Diehl (2022) demonstrate that posts emphasizing personal, experiential content are perceived as more genuine and receive higher engagement compared to posts that emphasize material acquisitions. This finding suggests that authenticity, as signaled through experiential cues, may be a key mechanism through which physical appeal enhances social media engagement.

To address the challenges of studying physical appeal at scale, we employ an AutoML-based AI approach to code and analyze a real-world dataset. We illustrate the application of our proposed estimator, which accounts for AI-coding errors, ensuring accurate and reliable inference from AI-coded data in large-scale marketing research. By examining the relationship between physical appeal and user engagement on Instagram at scale, we provide insights into the dynamics of social media interaction that can inform strategies for brands and influencers seeking to optimize their content for enhanced audience interaction on Instagram's visually driven landscape.

5.1 Data and Model

The foundational dataset for this study, collected by Kim, Jiang, and Wang (2021) and made available to us, focuses on brand-sponsored content on Instagram. This dataset is notable for its extensive scale, encompassing 1,601,074 posts that reference 26,910 distinct brand names, published over a six-year period from 2013 to 2019. It includes all posts that mention at least one brand name in the captions, as posted by 38,113 social media influencers with at least 1,000 followers. The average follower count of these influencers is 127,279. The dataset also includes key engagement metrics such as the number of followers of each influencer, the number of followers of the mentioned brand(s), the number of likes and comments on each post, and whether or not a post was sponsored (i.e., if the post either uses Instagram's branded content tool or includes one of the following hashtags: #ad, #sponsored, or #paidAD—widely used for sponsorship disclosure in influencer marketing). In sum, it provides a rich and diverse source for analyzing brand engagement on social media.

Despite the comprehensiveness of the dataset, it lacks explicit measures of physical appeal, which is the central construct of interest in our study. This limitation necessitates the development

of a method to quantify physical appeal from the available data, which we address through our proposed AI-based coding approach. This approach not only allows us to investigate our research questions by overcoming the limitations of previous studies but also demonstrates the utility of AI in augmenting existing datasets with new, theoretically relevant variables.

We conduct our analyses in two phases, both utilizing manually-coded measures of physical appeal for 5,000 randomly selected images. We limited the sample size in this study to ensure the feasibility of manually coding all images. This allows us to establish the efficacy of our estimators by comparing results from a "ground truth" benchmark—the estimates from the completely human-coded sample using canonical estimators—to those obtained from AI-coded data using various estimators (including the naive estimator and our proposed estimator). In other studies, it would be straightforward to apply our developed AutoML model to all images and all data points.

The manually-coded measures were developed by employing twenty independent coders who evaluated each image on a 7-point Likert scale assessing the extent to which it successfully makes a physical appeal; images judged not to make a physical appeal were coded as 0. In the first phase, we examine the relationship between physical appeal and two additional manually-coded variables: "engaging" and "exciting," which represent the extent to which a coder found a coded image engaging and exciting, respectively, on a 7-point Likert scale. We treat "engaging" and "exciting" as dependent variables, with physical appeal z_{phy} as the focal independent variable, and estimate:

$$\text{Engaging} = \delta_{\text{eng}} + z_{\text{phy}}\delta_{\text{eng, phy}} + \varepsilon_{\text{eng}}, \tag{6}$$

$$\text{Exciting} = \delta_{\text{exc}} + z_{\text{phy}}\delta_{\text{exc, phy}} + \varepsilon_{\text{exc}}. \tag{7}$$

While the above equations are estimated on each coded image, the second phase broadens our analysis, relating the success of a physical appeal in a social media post to virality measures such

as the number of likes and comments. Here, we estimate:

$$\text{Likes} = \delta_{\text{lik}} + z_{\text{ave}}\delta_{\text{lik, ave}} + z_{\text{dis}}\delta_{\text{lik, dis}} + D_{\text{spo}}\delta_{\text{lik, spo}} + x_{\text{inf}}\delta_{\text{lik, inf}} + x_{\text{brd}}\delta_{\text{lik, brd}} + \varepsilon_{\text{lik}}, \tag{8}$$

$$\text{Comments} = \delta_{\text{com}} + z_{\text{ave}}\delta_{\text{com, ave}} + z_{\text{dis}}\delta_{\text{com, dis}} + D_{\text{spo}}\delta_{\text{com, spo}} + x_{\text{inf}}\delta_{\text{com, inf}} + x_{\text{brd}}\delta_{\text{com, brd}} + \varepsilon_{\text{com}}. \tag{9}$$

where Likes and Comments are the number of likes and comments, respectively; D_{spo} indicates whether the post is sponsored; z_{ave} is the average of the coders' ratings of physical appeal; z_{dis} is the dispersion of the coders' ratings of physical appeal; x_{inf} represents the number of influencer's followers; and x_{brd} represents the number of the mentioned brands' followers. We include both z_{ave}, the average of the coders' ratings, and z_{dis}, the dispersion of the coders' ratings, to examine if, in addition to the presence of a successful appeal, the degree of agreement or disagreement among coders on the extent of the appeal plays a role. This corresponds to investigating whether not only the overall level of physical appeal but also the consistency of its perception across users influences engagement outcomes like likes and comments.

To investigate the implications of AI-coding, we apply AutoML-based AI systems; z_{ave} and z_{dis} are both manually-coded and AI-coded in Equations (8) and (9). To simplify the exposition and for tractability, we focus on the 5,000 social media posts that correspond to our sample of 5,000 images. This allows us to compare the canonical estimator given ground truth (i.e., the manually-coded measures), our proposed estimators given both AI-coded and manually-coded measures (for varying fractions of the data), and the canonical estimator given only AI-coded data. We restrict the sample in our analysis for interpretability and ease of exposition; our method is straightforward to scale to larger sample sizes and can also be adapted for 'online' analysis, where data must be analyzed as it is generated, as is now increasingly common in mobile marketing applications (such as through the use of Apple intelligence in the Apple ecosystem).

5.2 Impact of Physical Appeal on Engagement and Excitement

Table 4 reveals a strong positive relationship between physical appeal and both engagement and excitement. Specifically, a one-point increase in physical appeal is associated with a 0.530-point increase in engagement and a 0.550-point increase in excitement, both statistically significant at the 1% level.

Table 4: Physical Appeal on Engagement and Excitement

	Dependent Variable	
	Engaging	Exciting
Physical Appeal	0.530^{***}	0.550^{***}
	(0.002)	(0.002)
Constant	2.130^{***}	1.972^{***}
	(0.007)	(0.007)
Observations	100,000	100,000
R^2	0.476	0.487
Note: *p < 0.1; **p < 0.05; ***p < 0.01		

The substantial and similar effect sizes, coupled with the high R^2 values (0.476 for engagement and 0.487 for excitement), indicate that physical appeal consistently impacts multiple dimensions of user response. These findings align with prior research emphasizing the importance of visual content in fostering stronger emotional connections and higher levels of engagement on social media platforms, while extending previous insights from traditional marketing contexts to the emerging context of Instagram. Moreover, because engagement and excitement are known antecedents of social media virality, these findings suggest that physical appeal is likely to have significant effects on our dependent variables in the second phase of our analysis—likes and comments.

5.3 AI-Coding Errors in Physical Appeal

Figure 5 presents histograms of the coding errors—the differences between the manually coded and AI-coded measures of physical appeal—separately for sponsored and non-sponsored posts. Overall, we find a strong alignment between the manually coded and AI-coded measures, with

coding errors centered around zero and a very high correlation of 0.97.

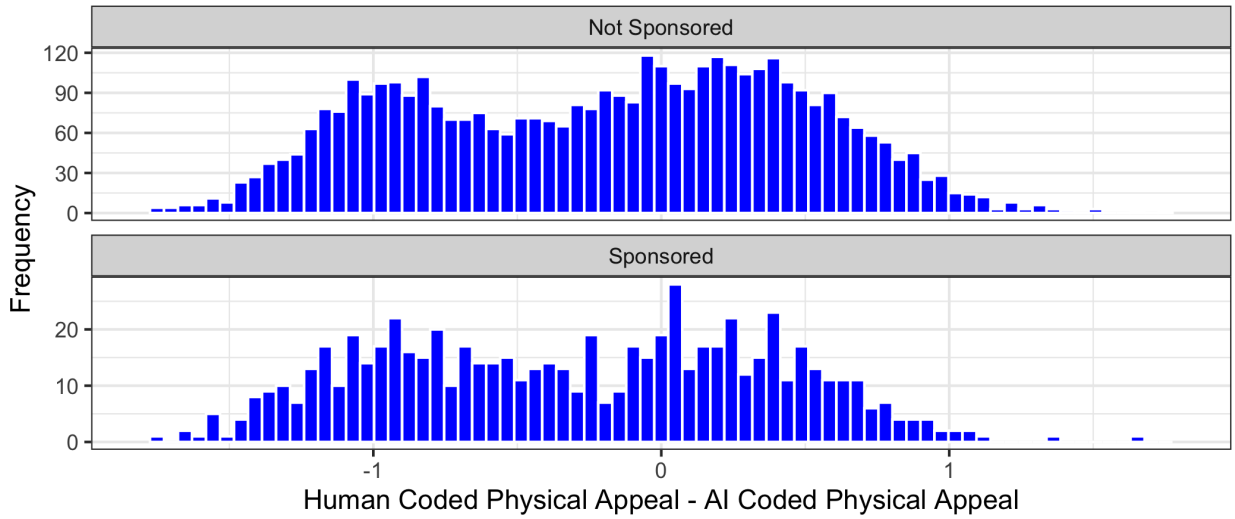


Figure 5: Histogram of AI-coding Errors in Coding the Success of Physical Appeals

However, we observe notable differences in the distributions of coding errors between sponsored and non-sponsored posts. For non-sponsored posts, the errors are more symmetrically distributed around zero, indicating consistent AI performance. In contrast, the errors for sponsored posts exhibit a more pronounced bimodal distribution. This suggests that specific features of sponsored content—such as the clothing or products featured—may be influencing the AI's assessment of physical appeal. For example, the AI may overestimate physical appeal when an influencer's appearance is enhanced by a sponsoring brand's products.

This is supportive of our claim that AI-coding errors may not be purely random; they may be systematically related to other features of the data, complicating their treatment in econometric models. Moreover, because the AI's predictions are influenced by complex interactions within the data, the nature of the errors is difficult to predict *a priori* or fully interpret *a posteriori*. Therefore, rather than attempting to explain these differences, we present an econometric approach that first identifies the AI-coding errors and then employs their statistical properties to ensure consistent inference, as we demonstrate in the next subsection.

5.4 Impact of Physical Appeal on Comments and Likes

To investigate the influence of AI-coding error on estimates and demonstrate the practical utility of our estimators in real-world applications, we construct five datasets with varying levels of manually-coded measures: 100% (all data manually-coded), 20%, 10%, 5%, and 0% (only AI-coded data). The 100% manually-coded dataset serves as our gold standard, where we apply the canonical estimator to obtain benchmark estimates. The datasets with 20%, 10%, and 5% manually-coded data showcase our proposed estimators' ability to correct for AI-coding errors using calibration subsamples. The 0% dataset represents a baseline scenario where coding error is ignored, simulating conditions where researchers lack access to manually-coded data and cannot apply our proposed estimators.

Table 5: Impact of Physical Appeal on Likes and Comments on Instagram

Model	Coefficients	Manual	20%	10%	5%	AI
Likes	Physical Appeal (Mean)	0.285^{***}	0.310^{***}	0.310^{***}	0.312^{***}	0.420^{***}
		(0.013)	(0.017)	(0.019)	(0.023)	(0.022)
	Physical Appeal (Dispersion)	-0.131^{***}	-0.240^{***}	-0.244^{***}	-0.255^{**}	-0.370^{***}
		(0.042)	(0.079)	(0.091)	(0.116)	(0.127)
	Sponsored Post	0.162^{**}	0.171^{**}	0.171^{**}	0.172^{*}	0.165^{**}
		(0.065)	(0.067)	(0.077)	(0.096)	(0.065)
	Number Influencer Followers	0.395^{***}	0.434^{***}	0.434^{***}	0.434^{***}	0.394^{***}
		(0.019)	(0.113)	(0.114)	(0.115)	(0.019)
	Number Brand Followers	-0.005^{**}	-0.005^{**}	-0.005^{**}	-0.005^{**}	-0.005^{**}
		(0.002)	(0.002)	(0.002)	(0.002)	(0.002)
Comments	Physical Appeal (Mean)	0.220^{***}	0.244^{***}	0.245^{***}	0.247^{***}	0.330^{***}
		(0.012)	(0.015)	(0.017)	(0.022)	(0.019)
	Physical Appeal (Dispersion)	-0.159^{***}	-0.265^{***}	-0.270^{***}	-0.281^{**}	-0.427^{***}
		(0.038)	(0.072)	(0.084)	(0.112)	(0.114)
	Sponsored Post	0.083	0.091	0.090	0.092	0.085
		(0.058)	(0.062)	(0.073)	(0.092)	(0.058)
	Number Influencer Followers	0.251^{***}	0.274^{***}	0.274^{***}	0.274^{***}	0.251^{***}
		(0.017)	(0.067)	(0.068)	(0.070)	(0.017)
	Number Brand Followers	-0.007^{***}	-0.007^{***}	-0.007^{***}	-0.007^{**}	-0.008^{***}
		(0.001)	(0.002)	(0.002)	(0.003)	(0.001)

Intercept suppressed. Standard errors in parentheses. $^{*}p<0.1$; $^{**}p<0.05$; $^{***}p<0.01$. Manual: canonical estimator on manually-coded data. 20% to 5%: proposed estimators; 20% to 5% of the data is manually-coded. AI: canonical estimator on AI-coded data.

Table 5 indicates that AI-coding error amplifies the estimated effect sizes in both models. In the likes model, for example, the coefficient on z_{ave} is 0.285 in the manually coded data and 0.420 in the AI-coded data, while the coefficient on z_{dis} is -0.131 manually and -0.370 when AI-coded. Similarly, in the comments model, the corresponding coefficients change from 0.220 to 0.330 and -0.159 to -0.427. In all four cases, the confidence interval for the AI-coded estimate does not include the manually coded estimate, underscoring how AI-coding imprecision can bias inference. Thus, if one were to rely on AI-coded data and canonical estimators alone, theories developed from these findings would likely be inaccurate relative to results derived from the more reliable, manually coded data.

By contrast, our proposed estimators—which use small subsamples of manually coded data to adjust for AI-coding errors—produce estimates that align much more closely with the manually coded benchmarks. In the likes model, for instance, the coefficient on z_{ave} when using 20%, 10%, and 5% manually coded data is 0.310, 0.310, and 0.312, respectively, compared to the AI-coded estimate of 0.420 and the manually coded benchmark of 0.285. In the comments model, the corresponding estimates (0.244, 0.245, and 0.247) also fall near the manually coded benchmark of 0.220, rather than the AI-coded estimate of 0.330. Estimates for z_{dis} show similar improvements, consistently returning to the confidence interval of the manually coded data. Hence, even in practical research contexts where large-scale manual coding is infeasible, our approach corrects for AI-coding biases and yields valid, reliable estimates.

Our results also reveal that sponsored posts receive more likes but not necessarily more comments: the coefficient on "Sponsored Post" is positive and significant in the likes model across all datasets but is nonsignificant in the comments model. This finding suggests that sponsorship increases post visibility, thereby encouraging users to click "like," yet may not prompt them to engage more deeply by commenting.

Furthermore, the coefficient on the influencer's number of followers is positive and significant in both models, indicating that influencers with larger followings typically garner more likes and comments. In contrast, the coefficient on the number of brands' followers is negative

and significant, implying that posts mentioning smaller or niche brands tend to elicit higher engagement than those mentioning major brands. One plausible explanation is that users perceive content featuring smaller brands as more authentic or novel, resulting in increased engagement. Conversely, larger brands may receive less organic engagement unless the posts are sponsored, consistent with our earlier findings.

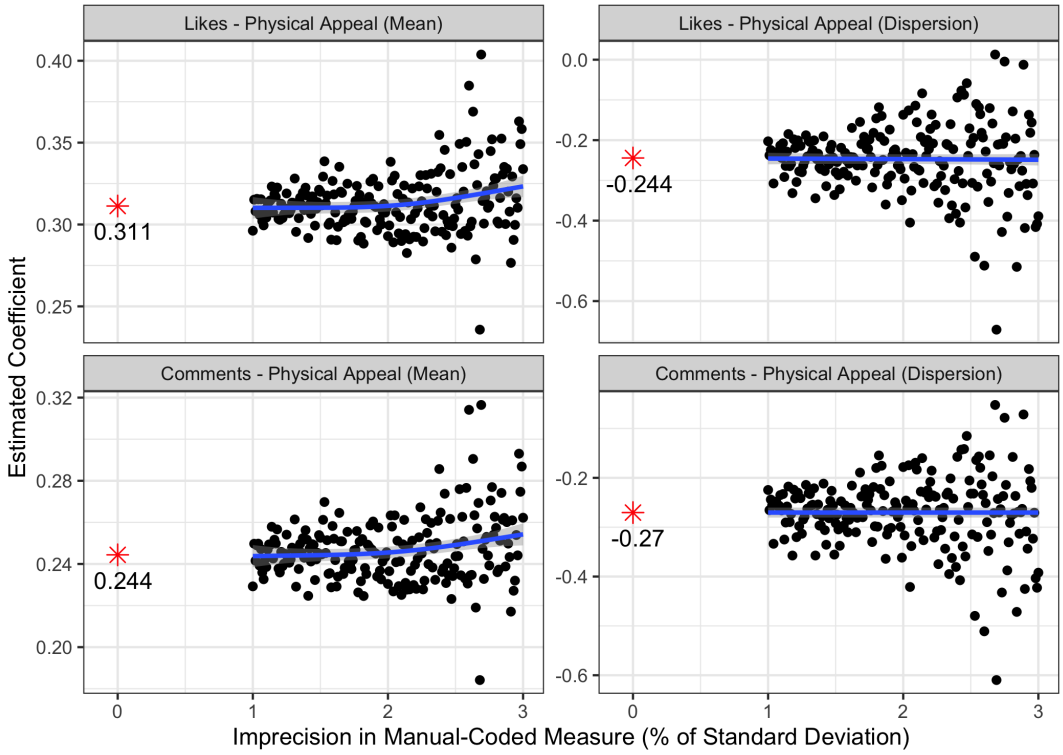
5.5 Manual-Coding Errors in Physical Appeal

To ensure the reliability of our manual coding process, we computed both the intraclass correlation coefficient (ICC) and Cronbach's alpha. The results indicated exceptional agreement among coders, with an average-measures ICC of 0.95 (95% CI: 0.95-0.96) and a Cronbach's alpha of 0.96 (95% CI: 0.96-0.96), demonstrating very high reliability in the manual coding of physical appeal.

Nevertheless, from a broader perspective, manual coding errors can still influence inference. To assess this possibility, we created datasets by systematically introducing varying degrees of imprecision into our manually coded measures of physical appeal. We focused on scenarios in which 20% of the data was manually coded—a proportion chosen to balance the logistical challenge of coding the entire dataset with the need for a sufficiently large calibration subsample to ensure accurate estimates.

Figure 6 presents our estimates for the two focal variables—the coefficients on the mean and dispersion of physical appeal—after incrementally increasing the standard deviation of artificially introduced noise from 1% to 200% of the original variables' standard deviation. For both variables, we plot the estimated coefficients (black points) from the likes and comments models and fit a GAM model (blue line) with standard errors (gray bands). We chose a GAM model out of an abundance of caution, as it is a nonparametric model that adapts to the potential curvature in the data. In this application (akin to the results in our simulations), a spline regression or even a linear regression would provide substantively similar results.

The GAM model smooths out sampling noise, producing an expected value for each level of coding imprecision. In line with a conservative assumption that our manually coded variables



Black points represent estimates. The x-axis shows imprecision as a percentage of the standard deviation of the original variable. Generalized Additive Model (GAM) fits are shown in blue, with standard errors in gray. Red asterisks denote extrapolated estimates corresponding to zero manual-coding error; the specific values of each extrapolate appear below its asterisk.

Figure 6: Implications of Imprecision in the Manually Coded Measures

contain significant measurement error—such that the coding-error standard deviation matches that of the original manual codes—we extrapolate to the scenario of zero measurement error (indicated by red asterisks).

Our analyses reveal that our proposed estimators are highly resilient to imprecision in the manual codes. While the raw estimates fluctuate due to sampling variance, the GAM fits remain relatively stable across imprecision levels ranging from 0% to 200% of the original variable's standard deviation. Moreover, the extrapolated estimates align closely with the coefficients obtained under precise manual coding.

This robustness arises because errors in the manually coded measures induce related variations in the AI-coded measures. Our estimators depend on manually coded data only for uncovering the correlation between AI-coding errors and other model variables. Since manual-coding imprecision

typically results from sampling variance and random coder-specific biases—both of which are orthogonal to predictor variables—its impact on these correlations, and thus on our proposed estimators, remains minimal.

6 Study: Humor as a Driver of Advertising Engagement

Engagement—the quantity and quality of user interaction with media, reflecting psychological involvement, social interaction, and immersion with a platform or its content—is widely recognized as a critical response evoked by advertisements (e.g., Olney, Holbrook, and Batra 1991; Pham, Geuens, and De Pelsmacker 2013). However, how humor specifically affects engagement in real-world data remains underexamined. Previous research primarily relies on "studies … performed in controlled laboratory settings, [that] are only mildly amusing, and [whose] effects may therefore differ from the effects of real-world advertisements" (Eisend 2009, p. 196).

Recent industry findings reinforce the importance of examining humor in a naturalistic context. Kantar, for instance, reports that although humorous advertisements demonstrate significantly higher expressiveness (+27%), involvement (+14%), and distinctiveness (+11%), only 33% of advertisements use humor. Adoption varies by channel: television (37%), YouTube (30%), and Facebook/Instagram (24%) (Media 2022). Notably, advertisements that deploy humor successfully benefit from heightened distinctiveness, emotional connectivity, persuasiveness, and brand equity, indicating substantial untapped potential for humor in advertising creativity (Jones 2023).

Addressing this gap, our study examines the influence of humor on viewers of real-world video advertisements, exploring whether and how humor in ads evokes engagement. We draw on data initially collected, and made available to us, by Hussain et al. (2017), comprising links to video advertisements on YouTube. We followed these links to download 1,610 ads that were still accessible—the remainder had been made publicly inaccessible.[2]

Twenty coders were assigned to evaluate humor in each advertisement using a 7-point Likert

[2]We manually examined the inaccessible links and found no systematic pattern related to the humor content of the ads or other variables relevant to our analysis. Likely reasons for removal include shifts in brand strategy, communication campaigns, or intellectual property rights concerns (e.g., expired rights for licensed music).

scale, where 1 corresponded to "not funny" and 7 to "very funny." Coders were randomly assigned to a subset of ads and did not know the study's objectives. Inter-coder reliability was assessed via the intraclass correlation coefficient (ICC) and Cronbach's alpha, both reflecting high agreement among coders (average-measures ICC of 0.82, 95% CI: 0.81-0.83; Cronbach's alpha of 0.87, 95% CI: 0.86-0.88), underscoring the reliability of the manual humor coding.

We also developed and applied an AI system to code humor. The AI model parsed each video into 1-second clips and distinct scenes, extracting and characterizing key features from each segment, and then using these successive segment-level characterizations to code the entire ad. In addition, we consolidated a categorical variable for each ad's category, derived from coders' open-ended descriptions. The ads in our sample predominantly promote Cars, Chips, Chocolate, Clothing, Electronics, Games, Media, Politics, Restaurants, Soda, and Vacation, with remaining ads grouped under "Other."

6.1 Coding Error in AI-coded Humor

From the coders' responses, we derived two aggregate measures of humor in each advertisement: the overall level of humor (z_{ave}), calculated as the average of the coded values, and the dispersion of coded values (z_{dis}), which captures the degree of agreement or disagreement among coders regarding how humorous the video is. The AI-coded measures exhibit strong correlations—0.90 for z_{ave} and 0.71 for z_{dis}—with the manually coded measures, providing strong evidence for the utility of AI in coding advertisements, while also highlighting the need for careful consideration of potential biases.

Figure 7 shows a violin plot of the AI-coding error in z_{ave}. The plot reveals systematic differences across advertisement categories. In consumer packaged goods categories such as chips, chocolate, and soda, coders consistently rated the advertisements as funnier than the AI did. Conversely, in more "serious" categories such as politics, where humor is less expected, the AI often rated the advertisements as funnier than the coders. These discrepancies highlight the potential for systematic bias in analyses that fail to account for coding error. To investigate the

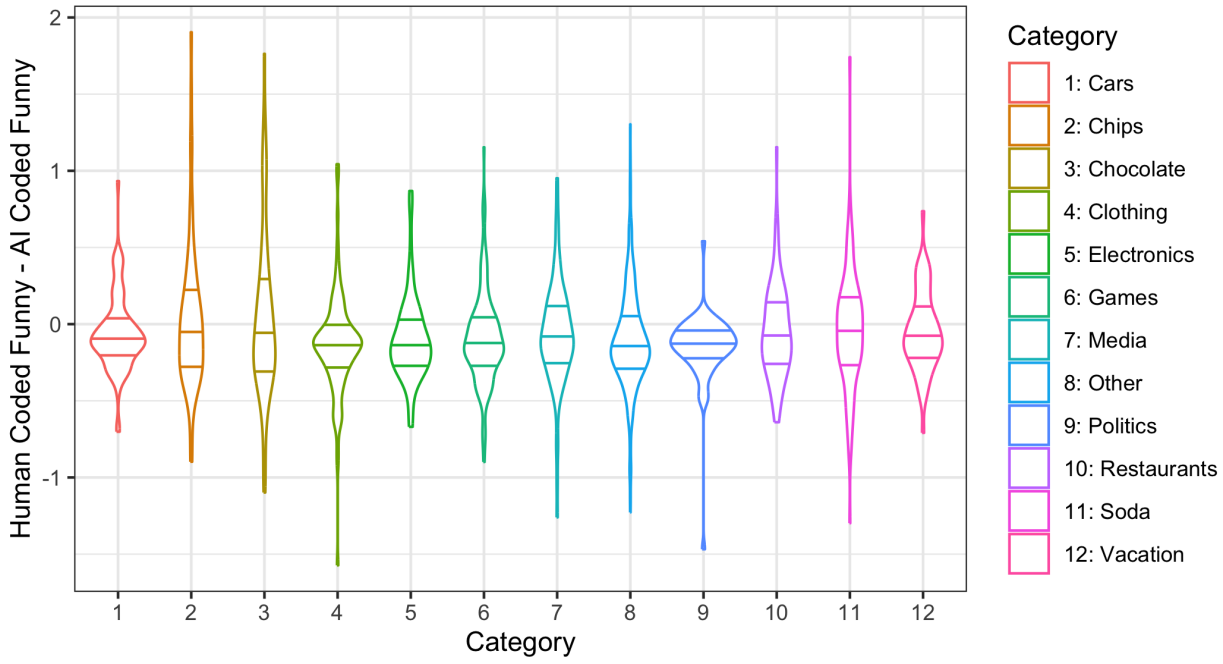


Figure 7: Violin Plot of AI-coding Error in Humor by Advertisement Category

Note: The violin plot depicts the distribution of AI-coding error—that is, the difference between manually coded and AI-coded values—across various advertisement categories. The width of each violin reflects the density of data points at different error levels, with horizontal lines indicating the 25th, 50th (median), and 75th percentiles.

consequences of these coding errors, we proceed to analyze the relationship between humor and engagement, comparing results obtained with and without correcting for AI bias.

6.2 Model and Results

Following our approach in the Instagram study, we construct five datasets with varying proportions of manually coded measures: 100% (all data manually coded), 40%, 30%, 20%, and 0% (only AI-coded data). We use a higher proportion of manually coded data than in the Instagram study due to the smaller total number of observations, the increased model complexity from including a categorical topic variable, and the need for sufficient data to ensure model stability and inference. We also obtained coders' ratings of engagement, defined as the ad's ability to attract and retain attention, a key factor in advertising persuasion. We estimate the following model and report the coefficients

β_1, β_2, and β_{3j} for $j = 1, \ldots, 12$:

$$\text{Engaging} = \beta_0 + \beta_1 z_{\text{ave}} + \beta_2 z_{\text{dis}} + \sum_j \beta_{3j} \text{Category}_j + \varepsilon. \tag{10}$$

Table 6: Regression Estimates for the Impact of Humor on Engagement, Using Varying Proportions of Manually-Coded and AI-Coded Data

Coefficient	Manual	40%	30%	20%	AI (Uncorrected)
Humor (Mean)	0.090^{***}	0.136^{**}	0.138^{**}	0.140^{**}	-0.033
	(0.029)	(0.054)	(0.058)	(0.070)	(0.030)
Humor (Dispersion)	-0.043	-0.126	-0.132	-0.143	0.224^{**}
	(0.047)	(0.195)	(0.209)	(0.257)	(0.087)
Chips	-0.233^{***}	-0.264^{***}	-0.264^{***}	-0.266^{***}	-0.161^{**}
	(0.081)	(0.085)	(0.089)	(0.100)	(0.082)
Chocolate	-0.181^{**}	-0.204^{**}	-0.205^{**}	-0.207^{**}	-0.120
	(0.078)	(0.083)	(0.087)	(0.098)	(0.079)
Clothing	0.075	0.068	0.068	0.068	0.072
	(0.072)	(0.073)	(0.076)	(0.082)	(0.072)
Electronics	-0.038	-0.047	-0.048	-0.048	-0.026
	(0.076)	(0.081)	(0.084)	(0.091)	(0.076)
Games	0.105	0.090	0.089	0.089	0.122^{*}
	(0.065)	(0.074)	(0.076)	(0.083)	(0.065)
Media	0.023	0.012	0.012	0.011	0.032
	(0.073)	(0.085)	(0.087)	(0.095)	(0.073)
Other	-0.080	-0.092	-0.092	-0.093	-0.067
	(0.056)	(0.062)	(0.064)	(0.070)	(0.057)
Politics	-0.387^{***}	-0.385^{***}	-0.385^{***}	-0.387^{***}	-0.399^{***}
	(0.082)	(0.080)	(0.082)	(0.089)	(0.082)
Restaurants	-0.250^{***}	-0.275^{***}	-0.275^{***}	-0.276^{***}	-0.212^{***}
	(0.075)	(0.075)	(0.079)	(0.089)	(0.075)
Soda	-0.116^{*}	-0.139^{*}	-0.139^{*}	-0.140	-0.077
	(0.070)	(0.076)	(0.079)	(0.086)	(0.071)
Vacation	0.233^{***}	0.233^{**}	0.233^{**}	0.232^{**}	0.234^{***}
	(0.085)	(0.093)	(0.096)	(0.104)	(0.085)

Intercept suppressed. Standard errors in parentheses. $^{*}p<0.1$; $^{**}p<0.05$; $^{***}p<0.01$. Manual: canonical estimator on manually coded data. 40% to 20%: proposed estimators (using calibration subsamples to correct for AI-coding error); 40% to 20% of the data is manually coded. AI (Uncorrected): canonical estimator on AI-coded data.

Table 6 presents our estimates. Our results indicate that correcting for AI-coding error is crucial for obtaining accurate estimates of the relationship between humor and engagement.

When we estimate the model using manually coded data, the coefficient on the mean of the humor ratings is positive and significant (0.090), ($p < 0.01$), confirming that funnier ads are perceived as more engaging. However, when estimated using only AI-coded data without correction, this coefficient becomes negative and nonsignificant (−0.033). Similarly, the coefficient on the dispersion of humor ratings—which captures the extent of disagreement among viewers about humor in the ad—is negative and nonsignificant in the manually coded data (−0.043), suggesting that greater disagreement does not affect engagement levels. In contrast, in the AI-coded data without correction, this coefficient is positive and significant (0.224), ($p < 0.05$).

Thus, despite the very strong correlations (0.90 and 0.71) between our manually coded and AI-coded measures, we find evidence of both Type I errors (incorrectly rejecting a true null hypothesis) and Type II errors (failing to reject a false null hypothesis) in the AI-coded analysis without correction. Turning to the category-specific effects, we find that ads for Vacation are associated with higher engagement, while ads for more mundane products such as Chips, Chocolate, and Soda are associated with lower engagement. While the estimates of the category effects are broadly consistent, the nonsignificant coefficient on Chocolate in the AI-coded analysis further highlights the potential for AI-coding errors to distort the estimated relationships.

Our proposed estimator effectively corrects for these biases, yielding results that closely align with the manually coded estimates. The estimates from the 40% manually coded subsample are robust and closely align with those from the 100% manually coded scenario in both direction and magnitude. The results from the 30% and 20% subsamples demonstrate some instability, likely due to the inclusion of a categorical variable, which requires sufficient data per category for precise effect size estimation and to avoid numerical instability; a 20% subsample (320 observations) may be too small in this context. Overall, however, our results illustrate the efficacy of our method in correcting for AI-coding errors and obtaining reliable estimates even with partial manually coded data. They strongly support the validity and practical applicability of our proposed estimators in real-world settings.

7 General Discussion

Machine learning (ML) has become a critical tool across various fields for analyzing causal research questions—including creativity and idea generation (Toubia and Netzer 2017), education (Zhou et al. 2021), financial markets (Bao and Datta 2014), political science (Egami et al. 2018), and social media content analysis (Zhong and Schweidel 2020). In these applications, ML is employed to 'structure' unstructured data by converting complex, non-numerical inputs into concise numerical and categorical measures.

For instance, Toubia, Berger, and Eliashberg (2021) examine the influence of narrative structure on success by introducing novel quantitative metrics—speed, volume, and circuitousness—that capture essential determinants of semantic progression in discourse. These metrics offer richer insights than traditional measures such as star power and budget, which have historically dominated studies on movie success (Mukherjee and Kadiyali 2011). However, operationalizing these variables required applying ML models capable of effectively extracting and quantifying relevant features. Our proposed use of AI follows the same principles and objectives: because unstructured data lack a predefined, rigid model, their informative elements must be extracted—a role that AI is well-suited to fulfill.

Substantively, we address a central challenge in consumer research: extending established experimental findings on emotional appeals—such as humor and physical attractiveness—to large-scale, real-world settings. While laboratory-based studies have long suggested that emotional appeals significantly influence consumer attitudes and behaviors, empirical validation in naturalistic contexts has been limited by the high cost of manual data coding. In response, we develop a framework that leverages modern AI systems to code unstructured data automatically and then applies new econometric estimators to correct for AI-coding errors.

We make three principal contributions. First, we demonstrate that state-of-the-art AI systems, particularly those employing AutoML, can effectively capture complex constructs such as humor and physical appeal in social media posts and video advertisements. In our Instagram and YouTube

applications, AI-coded variables correlate strongly with human judgments—up to 0.97 for physical appeal and 0.90 for humor—illustrating the promise of automated coding at scale. Second, we highlight the pitfalls of directly substituting AI-coded variables into econometric models. Through simulations and empirical analyses, we show that AI-coding errors can bias effect estimates by as much as 40–50%, sometimes even reversing their sign. These biases arise because AI-coding errors are often correlated with other covariates or instruments. Our analyses underscore how failing to address this issue can produce erroneous conclusions, including both Type I (false positive) and Type II (false negative) errors. Third, we introduce novel estimators that exploit a small calibration subsample of manually coded data to correct for AI-coding errors in the full sample. These estimators, whether in single-equation or system-of-equations contexts, consistently yield parameters close to ground truth—even when fewer than 10% of observations are manually coded. Our applications—Instagram posts featuring physical appeals and YouTube video ads incorporating humor—illustrate how this dual approach of AI coding plus bias correction provides large-scale insights without sacrificing econometric rigor.

Beyond these substantive contributions, our findings have wide-reaching implications. For theory development, the ability to measure emotional appeals in naturalistic, unstructured data opens new avenues for enhancing the ecological validity of a wider range of research questions(Van Heerde et al. 2021)—examining how consumers respond to marketing stimuli in different cultural contexts (e.g., comparing humor responses in the US vs. Japan), product categories (e.g., high- vs. low-involvement goods), and media platforms (e.g., TikTok vs. LinkedIn). For practitioners, the results underscore the importance of robust measurement to ensure the ecological validity of their insights and decisions. We show that, for instance, the impact of humor on ad engagement can be understated or overstated if AI-coding errors go uncorrected—potentially leading advertisers to misallocate resources when planning campaigns. More broadly, the methodology can be adapted to a diverse range of marketing constructs that emerge in text, images, audio, or video, from sentiment in customer reviews to brand personality in influencer posts.

Looking ahead, several research directions stand out. As multimodal AI continues to ad-

vance, holistic models—drawing simultaneously on text, images, and audio—could code complex emotional dimensions even more accurately. Real-time AI coding may enable researchers to adapt content dynamically, refining advertising strategies on the fly. Additionally, integrating our bias-correction strategies with privacy-preserving methods—such as federated learning—could facilitate large-scale, ethical data analyses even where raw data cannot leave user devices.

While our study provides a robust framework for leveraging AI to study emotional appeals in large-scale, real-world data, it is important to acknowledge several limitations. First, the effectiveness of our bias-correction method depends on the quality and representativeness of the manually coded calibration subsample. While we demonstrate strong results with subsamples of 5–20%, smaller or non-randomly selected subsamples could potentially limit the accuracy of the estimators, particularly in contexts with low prevalence of the target construct or high heterogeneity in its expression. Second, our empirical analyses focused on humor and physical appeal on Instagram and YouTube. Future research should investigate the applicability of our approach to a broader range of emotional appeals and across different social media platforms. Additionally, while we found strong correlations between AI-coded and manually coded variables, it is important to acknowledge the potential for cultural and contextual biases in AI models. Future research could examine how these biases manifest and develop methods for mitigating their impact, particularly in cross-cultural studies. Finally, our study primarily focused on short-term engagement metrics. Future research should investigate the impact of emotional appeals on longer-term outcomes, such as brand attitudes and purchase behavior, to gain a more comprehensive understanding of their effectiveness.

In sum, we provide a practical roadmap for reliably incorporating AI-coded variables into large-scale marketing studies. By striking a balance between the scalability of automated coding and the precision of manual annotation, we hope our approach helps researchers and practitioners unlock deeper insights into how emotional appeals function in the rich—and rapidly evolving—landscape of contemporary consumer marketplaces.

WEB APPENDICES

A Web Appendices: Overview and Roadmap

This web appendix provides the technical foundations and implementation details for the AI-driven methodology presented in *"Emotional Appeals as Drivers of Social Media and Advertising Engagement in Real-World Marketplaces: Using AI to Code Variables in Consumer Research."* While the main paper focuses on the purpose, application, and empirical validation of the proposed approach, this appendix offers the complete technical details necessary to understand, replicate, and extend the methodology. This includes derivations, estimators, implementation steps, and the simulation code used to implement the models described in the main manuscript.

Our methodology addresses a core challenge in consumer research: leveraging the rich, unstructured data available from real-world sources while maintaining the rigor necessary for causal inference. Traditional consumer research often relies on carefully designed surveys or experiments, which offer a high degree of control. However, these approaches may not fully capture the complexity of consumer behavior in naturalistic settings. In contrast, the abundance of unstructured data from sources such as social media, online reviews, and digital advertising platforms offers a unique opportunity to study consumer behavior at scale and in context.

To bridge the gap between controlled experiments and real-world data, we integrate AI-coded variables, derived from unstructured data, into econometric models. These automatically generated measures serve as proxies for marketing constructs that are not directly observable in the raw data but are crucial for understanding consumer behavior. For example, in the main paper's simulation studies, AI models code the degree of nostalgia evoked by advertisements—a construct shown to influence consumer engagement and brand attitudes in laboratory settings but challenging to measure at scale with real-world data. By incorporating these AI-coded variables, we can examine the impact of these constructs on key marketing outcomes while accounting for potential biases introduced by AI-coding errors.

The appendix begins with an implementation guide offering a step-by-step walkthrough for: (1) obtaining and preparing unstructured marketing data, (2) manually coding a calibration subsample, (3) developing AI models using Automatic Machine Learning (AutoML), (4) estimating key statistical properties in the calibration subsample, and (5) integrating AI-coded variables into econometric models, including addressing imprecision in manually coded measures. Specifically, it details how to acquire data (e.g., from social media platforms, online reviews, and advertising databases), select a representative calibration subsample, and establish clear coding protocols to ensure inter-coder reliability. It then demonstrates how to leverage AutoML tools, such as Keras Tuner and AutoKeras, to train custom AI models for coding constructs like nostalgia appeals. Finally, it explains how to combine AI coding of the full dataset with the statistical properties estimated from the calibration subsample to achieve robust and consistent econometric estimation. The guide is designed to be accessible to researchers with varying levels of technical expertise, offering practical recommendations for implementation in real-world research settings.

A key contribution of our research is the development and application of bias-correction techniques to ensure valid inference when using AI-coded variables. While AI models provide a scalable solution for coding marketing constructs from unstructured data, they introduce systematic errors that can bias econometric estimates. Our research presents an inference framework for identifying, quantifying, and correcting these biases, enabling researchers to integrate AI-coded variables into econometric models without compromising statistical validity.

The next two sections develop this framework. The first examines single-equation models, such as those estimated using ordinary least squares (OLS) or two-stage least squares (2SLS). It derives the bias introduced by AI-coding errors—particularly when those errors are correlated with other regressors—and demonstrates that these errors can lead not only to attenuation but also to amplification of effect sizes. This section then presents novel bias-corrected estimators, identifies key conditions for identification, and provides a detailed algorithm for practical implementation, including bootstrap procedures for standard errors.

The analysis is then extended to system-of-equations models, such as three-stage least squares

(3SLS), which are commonly used to capture interdependencies and correlated errors across multiple marketing outcomes. We develop the bias-correction framework for multi-equation models, derive how AI-coding errors can propagate across equations, and introduce estimators that account for these biases. This section also provides guidance on estimating covariance structures, verifying identification, and executing bootstrap-based inference.

The penultimate section addresses the issue of imprecision in manually coded measures. Even when human coders are used to establish ground truth, their ratings may contain variability. We describe estimators that adapt the Simulation-Extrapolation (SIMEX) estimator to correct for human-coding imprecision and clarify the implications of imprecision in manually coded measures on parameter estimates and inference. A step-by-step algorithm is included, along with a discussion of the trade-offs between coding precision and dataset scale.

The final section of this web appendix provides the complete R code for the simulation studies that validate our proposed bias-corrected estimators. These simulations generate synthetic data mimicking real-world marketing scenarios with AI-coded variables, allowing for a controlled assessment of estimator performance under various conditions, including endogeneity, AI-coding errors in both regressors and instruments, and imprecision in the manually coded calibration data. Separate code implementations are provided for single-equation models, system-of-equations models, and a robustness analysis that incorporates the SIMEX-based method to address manual coding error.

B Implementation Guide to Using AI for Variable Coding in Consumer Research

This section provides a practical, step-by-step guide to implementing our proposed methodology, enabling researchers to analyze large-scale, unstructured marketing data using AI. It covers the following key steps:

1. **Obtaining and Preparing Data for AI Coding:** This subsection outlines the process of obtaining and preparing data for AI coding. It describes how to collect unstructured data from relevant sources (e.g., social media platforms, online forums, and advertising databases) and how to select and manually code a calibration subsample. Emphasis is placed on establishing clear coding protocols and ensuring inter-coder reliability. The subsection concludes with a discussion of data preprocessing techniques to maintain data quality and compatibility with AI models.

2. **Developing Custom AI Models with AutoML:** This subsection discusses the use of AutoML for developing custom AI models to code marketing-relevant variables. AutoML automates key steps in AI model development, making it accessible to researchers without extensive programming expertise. We present options for different AutoML platforms and provide guidance on selecting the most appropriate one. The subsection then describes the process of training, validating, and deploying AI models, highlighting the role of the manually coded calibration subsample in model evaluation.

3. **Estimating Moment Matrices in the Calibration Subsample:** This subsection details the estimation of moment matrices in the calibration subsample, explaining their role in capturing the statistical properties of AI-coding errors and their relationships with other model variables. We provide step-by-step instructions for computing key moment matrices, including the cross-product matrices of AI-coded regressors, instruments, and

coding errors. We also discuss how to assess identification conditions to ensure the validity of the estimators.

4. **Estimating Econometric Models with Bias Correction:** This subsection presents our bias-corrected estimators for both single-equation and system-of-equations models. We provide detailed instructions for implementing these estimators in standard statistical software, incorporating the estimated moment matrices to adjust for AI-coding errors. We also discuss hypothesis testing, inference, and model diagnostics in the context of AI-coded variables.

5. **Addressing Imprecision in Manually Coded Measures:** This subsection addresses the issue of imprecision in manually coded measures and its impact on accuracy. We introduce a SIMEX procedure to correct for biases introduced by manual coding errors. This subsection provides a detailed algorithm for implementing the SIMEX procedure, including guidance on generating synthetic datasets with varying levels of imprecision and extrapolating to the case of no manual coding error.

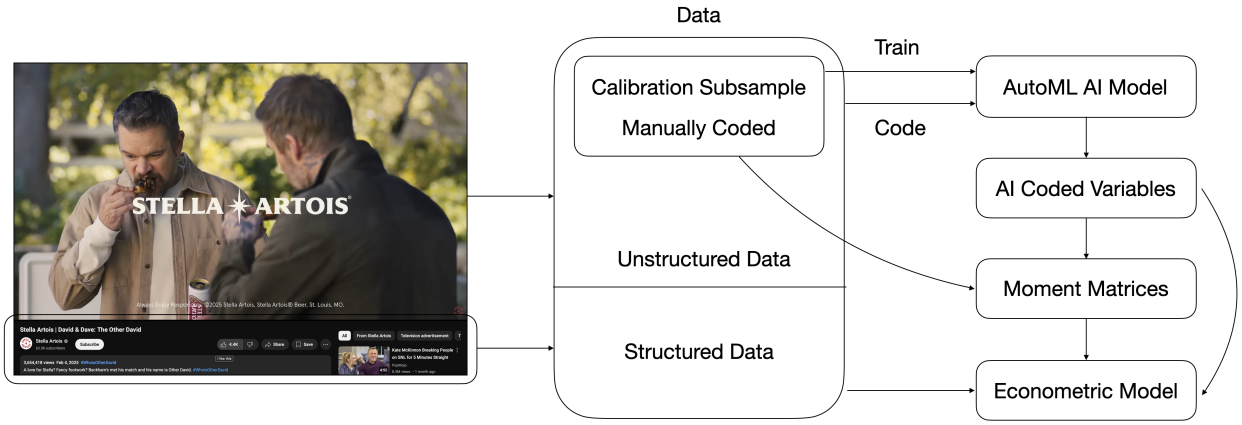


Figure 8: General Workflow in AI-Facilitated Marketing Research

Note. The left side shows a YouTube page consisting of unstructured data (the video) and structured variables (e.g., brand, likes, views). A subset of these videos is manually coded for the focal variable (e.g., humor). The resulting labels train the AutoML AI model, which then codes the complete data. The difference between AI predictions and the manually coded values in the calibration subsample is used to form error-correction moment matrices. These matrices and the structured data are then fed into an econometric model (e.g., estimating the effect of humor on likes), producing bias-corrected inference.

Figure 8 illustrates our proposed process. On the left is the YouTube page of a recent advertisement. At the top is the video advertisement itself, representing the unstructured data that must be coded. Below are several key pieces of structured information, including the channel hosting the advertisement (i.e., the brand being advertised), the number of likes the video has received, and the number of views. In this example, we treat these variables as structured information; in other studies, the data might include additional independent and dependent variables.

The right side of the figure illustrates the workflow. First, both the video advertisement (unstructured data) and the structured information are recorded (Step 1). A subset of these video advertisements is then selected as a calibration subsample and manually coded for the focal independent variable—for instance, humor (Step 2). The manually coded measures in the calibration subsample train the AutoML AI model, which then codes the remaining video advertisements, thereby scaling the manual-coding process (Step 3). The difference between the AI model's predictions and the manually coded values in the calibration subsample is used to compute the error-correction moment matrices, following formulas provided in Web Appendix §C, D, and E (Step 4). Finally, the structured data, the AI-coded measures, and the moment matrices feed into an econometric model in which the dependent variable of interest (e.g., number of likes) is regressed on both the other structured independent variables (e.g., the advertised brand) and the AI-coded variables (Step 5).

To illustrate this process, we use a running example throughout this guide: a *hypothetical* study examining the impact of nostalgia appeals in advertising on consumer engagement. This example mirrors the simulation studies presented in the main manuscript, providing a concrete illustration of the steps involved in applying AI coding to marketing research. Specifically, we envision an experiment in which participants are exposed to advertisements that vary in their use of nostalgic cues. The advertisements are sourced from a large-scale dataset of real-world social media posts and video ads, ensuring ecological validity. While distinct from the simulation studies used to validate the methodology in the main paper, this hypothetical example is set in the same context to reinforce key concepts. We revisit this example in each subsection, providing concrete

steps.

In the context of this example, step 1 would correspond to collecting both the unstructured data (the advertisements or their descriptions) and the structured data (any other measures such as engagement metrics). Step 2 would involve coding a calibration subsample of the advertisements manually. Step 3 involves training the AI to code the advertisements. Step 4 would involve recovering the moment matrices, and step 5 would involve estimating the econometric models in the complete data. Below, we now describe each of these steps in detail, presenting Python code (using libraries like `scikit-learn`, `Keras`, and `AutoKeras`) that can be readily adapted for use.

B.1 Obtaining and Preparing Data for AI-Facilitated Analysis

This subsection outlines the initial procedures for acquiring and preparing data within the proposed AI-facilitated research framework. These procedures encompass the collection of raw data and the establishment of a robust foundation for subsequent AI model training and bias correction.

A dataset in this context typically comprises two primary components: unstructured and structured data. These are defined as follows:

- **Unstructured Data:** This constitutes the core content from which marketing constructs of interest (e.g., nostalgia, humor, physical appeal) are to be extracted. Examples include images from Instagram posts, text from product reviews, or video streams from advertisements. In the running nostalgia example, this encompasses the advertisement content itself (images, videos, and accompanying text). This data is considered "unstructured" because it lacks predefined numerical variables representing these constructs; the AI model will be tasked with extracting this information.

- **Structured Data:** This encompasses readily available, quantifiable information associated with the unstructured content. Examples include the number of likes or comments on a social media post, the view count of a video, the price of a product, or demographic data

pertaining to a user. In the running nostalgia example, this might include the number of likes and comments on an Instagram post featuring the advertisement. These variables are amenable to direct use as dependent, independent, or control variables within econometric models.

Ad ID	Unstructured Content	Likes	Comments
2001	Image of a vintage product with retro design	1500	300
2002	Video ad featuring a historical event	2800	500
2003	Text post referencing a popular past trend	900	180

Table A7: Example of the raw data structure before AI coding.

Table A7 illustrates a sample dataset for studying nostalgia in advertising. Each advertisement is associated with a unique identifier (Ad ID), unstructured content, and structured data (engagement metrics such as likes and comments). The Ad ID facilitates data linkage and manipulation. The unstructured content contains embedded variables that require coding, while the engagement metrics capture observed consumer responses, enabling regression analysis of engagement on variables coded from the unstructured content. This table structure is illustrative and can be expanded to include additional data and variables relevant to a specific research question.

Incorporating AI-coded variables into econometric models introduces analytical challenges. Unlike traditional survey or experimental data, where independent variables are often directly observed or manipulated, AI-coded variables are *inferred* from complex, unstructured data. This inference process introduces the potential for coding errors, which may be correlated with other model variables, potentially biasing estimates. For example, an AI model trained to detect nostalgia in advertisements might exhibit varying accuracy depending on specific historical references, image quality, or the presence of certain keywords. If these factors are also correlated with engagement metrics (e.g., ads referencing more popular historical events receiving more likes, regardless of nostalgia intensity), the AI-coding errors will be correlated with the regressors, violating the assumptions of standard econometric estimators. Consequently, rigorous attention to potential biases is essential for valid inference.

To address these potential biases, our proposed methodology incorporates a calibration subsample. This subsample comprises a portion of the data that is *manually* coded by human annotators, providing "ground truth" measures of the constructs of interest. The calibration subsample serves three crucial purposes: (1) training the AI models used for automated coding, (2) estimating the statistical properties of AI-coding errors to enable bias correction, and (3) providing a benchmark for evaluating AI model performance.

Below, we first describe data sources that may prove useful in such studies. Then we discuss the selection and coding of the calibration subsample, the quality and representativeness of which are critical: if certain product categories, advertising formats, or demographic contexts are systematically excluded, the AI model and subsequent bias-correction process may yield inaccurate estimates.

B.1.1 Data Sources

The initial step involves obtaining relevant unstructured and associated structured data from sources reflecting real-world consumer behavior and appropriate for the research question. Marketing researchers can draw from a wide array of data sources, each offering unique insights into consumer preferences, attitudes, and actions. Some common sources include:

1. **Social Media Platforms:** Platforms like Instagram, Facebook, Twitter, TikTok, and YouTube provide a wealth of user-generated content, including text posts, images, videos, and comments. These platforms offer valuable insights into consumer engagement, brand sentiment, and the impact of marketing campaigns. For instance, researchers might collect Instagram posts related to a specific brand or product to analyze the use of nostalgia appeals in user-generated content.

2. **Online Reviews and Forums:** Websites like Amazon, Yelp, TripAdvisor, and Reddit host vast amounts of consumer reviews and discussions. These sources provide rich qualitative data on consumer experiences, product perceptions, and brand evaluations. Researchers can

scrape and analyze reviews to understand the factors driving customer satisfaction, identify unmet needs, or assess the effectiveness of marketing communications.

3. **Advertising Databases:** Platforms such as Kantar, Ad Age, and YouTube's Ad Leaderboard offer access to large collections of advertisements across various media channels. These databases can be used to study the prevalence and impact of different advertising strategies, including the use of nostalgia appeals, humor, and physical attractiveness.

4. **E-commerce Websites:** Online retailers like Amazon, eBay, and Etsy provide detailed product information, customer reviews, and sales data. Researchers can analyze product descriptions, images, and customer feedback to understand how product attributes and marketing messages influence consumer choices.

5. **News and Media Outlets:** Online news articles, blogs, and other media sources (e.g., LexisNexis, Factiva) can provide valuable contextual information about market trends, brand perceptions, and public opinion. These sources can be particularly useful for studying the impact of external events on consumer behavior or for tracking the evolution of brand narratives over time.

The choice of data sources depends on the specific research question and the constructs of interest. For example, a study on the impact of nostalgia in video advertising might focus on YouTube and advertising databases, while a study on brand authenticity might draw from social media platforms and online forums.

B.1.2 Calibration Subsample Selection

Following data source identification, the next step is to select a calibration subsample for manual coding. This subsample serves three crucial purposes: (1) training the AI models used for automated coding, (2) estimating the statistical properties of AI-coding errors to enable bias correction, and (3) providing a benchmark for evaluating AI model performance.

The selection of the calibration subsample should be guided by the following principles:

- **Representativeness:** The subsample should be representative of the full dataset in terms of the key constructs and variables of interest. For example, if the study focuses on the impact of nostalgia in advertising, the subsample should include advertisements that span the full range of nostalgia intensity, from not nostalgic to very nostalgic. Similarly, if the study examines multiple product categories or cultural contexts, the subsample should include examples from each relevant category or context.

- **Randomness:** To minimize selection bias, the subsample should be randomly drawn from the full dataset. Random sampling ensures that each observation in the full dataset has an equal probability of being included in the subsample, increasing the likelihood that the subsample accurately reflects the characteristics of the full sample.

- **Stratification (Optional):** In some cases, it may be beneficial to use stratified random sampling to ensure adequate representation of specific subgroups or categories within the data. For example, if the study examines the impact of nostalgia appeals across different product categories, strata may be created for each category and observations randomly sampled within each stratum. This approach ensures that each category is represented in the calibration subsample in proportion to its prevalence in the full dataset.

- **Size:** The size of the calibration subsample depends on several factors, including the complexity of the constructs being coded, the desired level of precision, and the available resources for manual coding. As a general rule, larger subsamples provide more reliable estimates of AI-coding error properties and enable more accurate training of AI models. However, researchers must balance the benefits of a larger subsample against the increased costs and time required for manual coding. In our empirical studies, subsamples comprising 5-20% of the full dataset were found to strike a good balance between accuracy and feasibility. These findings may be used as a rule-of-thumb when determining sample size.

B.1.3 Manual Coding Procedures

The manual coding of the calibration subsample establishes the "ground truth" against which the AI models are trained and validated. To ensure the reliability and validity of the manually coded data, researchers should follow best practices for human annotation. It is important to note that even with careful procedures, some degree of imprecision may remain in the manually coded data, particularly for subjective constructs. This potential for imprecision is addressed later using the SIMEX procedure.

1. **Develop a Coding Protocol:** A detailed coding protocol should be developed to guide the manual coding process. The protocol should provide clear definitions of the constructs being coded, along with specific criteria and examples for each level of the coding scale. For example, if coding for nostalgia, the protocol might define different types of nostalgia appeals (e.g., personal, historical, cultural) and provide examples of each type. The protocol should also specify the unit of analysis (e.g., the entire advertisement, individual scenes, specific characters) and the coding scale to be used (e.g., a 7-point Likert scale).

2. **Train Coders:** Human coders should undergo thorough training before beginning the coding process. The training should include an overview of the research objectives, a detailed explanation of the coding protocol, and practice coding with example data. During training, coders should have the opportunity to ask questions, clarify any ambiguities in the protocol, and calibrate their judgments with those of other coders.

3. **Independent Coding:** Each observation in the calibration subsample should be independently coded by multiple coders (in our experiments, up to 20 coders). Independent coding helps to reduce individual biases and provides a measure of inter-coder reliability. Coders should not be aware of each other's ratings during the coding process to avoid influencing each other's judgments.

4. **Assess Inter-Coder Reliability:** After the coding is complete, the level of agreement among

coders should be assessed using appropriate statistical measures, such as Krippendorff's alpha, Cohen's kappa, or the intraclass correlation coefficient (ICC). High levels of inter-coder reliability (e.g., alpha > 0.7) indicate that the coding protocol is clear and that the coders are applying it consistently. If reliability is low, researchers may need to revise the coding protocol, provide additional training, or collect additional data.

5. **Resolve Discrepancies:** When there are disagreements among coders, a procedure for resolving them should be implemented. Common approaches include taking the mean or median of the coders' ratings as the final score. Alternatively, a consensus-based approach may be used, where coders discuss their ratings and reach a joint decision. In some cases, it may be necessary to collect additional ratings from expert coders or to exclude observations with high levels of disagreement.

B.2 Developing Custom AI Models with AutoML for Coding Consumer Research Variables

With the calibration subsample manually coded, the full dataset acquired, and initial data cleaning performed, the next step is to develop custom AI models that can automatically code the marketing constructs of interest—in our running example, nostalgia appeals. While pre-trained AI models exist for some common tasks (e.g., object recognition, sentiment analysis), they may not be directly applicable to specialized marketing constructs or the specific nuances of a particular research question. Therefore, researchers often need to develop custom AI models tailored to their specific research questions and datasets.

Traditionally, developing custom AI models required substantial expertise in machine learning, including knowledge of model architectures, optimization algorithms, and software frameworks. However, AutoML (Automatic Machine Learning) significantly simplifies this process, making AI development accessible to researchers with limited programming experience. AutoML automates the end-to-end process of applying machine learning, from data preprocessing to model

deployment. Specifically, it systematically searches for the best-performing model architecture and hyperparameters, minimizing the need for manual, and often tedious, trial-and-error. This automation not only increases efficiency but also allows AutoML systems to explore a far broader range of possible architectures and configurations than a human researcher could reasonably manage, often leading to models that outperform those designed manually.

Several AutoML frameworks and platforms are available, each with its own strengths and weaknesses. Some popular options include:

- **Amazon SageMaker Autopilot:** A fully managed service that automatically builds, trains, and tunes the best machine learning models based on the data.
- **Auto-Sklearn:** An automated machine learning toolkit and a drop-in replacement for a scikit-learn estimator.
- **Google Cloud AutoML:** A suite of machine learning services that enables developers with limited ML expertise to train high-quality models specific to their business needs.
- **H2O.ai:** An open-source platform that provides a range of machine learning algorithms and AutoML capabilities.
- **Keras Tuner:** An easy-to-use, distributable hyperparameter optimization framework that streamlines hyperparameter search. This is a good option for researchers already familiar with the Keras/TensorFlow ecosystem.
- **TPOT (Tree-based Pipeline Optimization Tool):** An open-source Python library that uses genetic programming to optimize machine learning pipelines.

For this guide, we use Keras Tuner for hyperparameter optimization and AutoKeras for end-to-end AutoML. Both are open-source, well-documented, and integrate seamlessly with the popular TensorFlow/Keras deep learning framework. Keras Tuner provides more granular control over the model architecture, allowing for fine-tuning, while AutoKeras automates a larger portion of the process, including aspects of feature preprocessing and model selection. The optimal choice between these two will depend on the researcher's level of expertise and the specific requirements of the project.

B.2.1 Preparing Data for AutoML

Before training AutoML models, the data must be preprocessed to ensure its quality and compatibility with the chosen AutoML platform. While AutoML automates many aspects of model building, it still requires data to be in a specific, numerical format. This section builds upon the general data preparation steps outlined previously, focusing specifically on the requirements for using AutoML. It's crucial to remember that the order of these steps is important; specifically, data splitting should *always* precede data transformation.

1. **Feature Extraction:** This stage focuses on transforming *unstructured* data (images, text, audio) into *numerical* representations that the AI models can process. This is necessary because, unlike structured data (e.g., a table of customer demographics), unstructured data doesn't have predefined numerical features. The specific techniques used depend on the nature of the data. For instance:

    - **Image Processing:** For image data, common techniques include resizing and cropping images to a uniform size. For example, in our running example of coding nostalgia in video ads, all images might be resized to 224x224 pixels. Color normalization adjusts the color balance across images, ensuring consistency. Edge detection, often performed using Sobel operators (available in libraries like OpenCV[3]), highlights boundaries and shapes within images. Object recognition, leveraging pre-trained models like VGG16 in Keras[4] , can identify specific objects present in the images. In the nostalgia example, this might involve detecting vintage cars or clothing styles.
    - **Natural Language Processing (NLP):** For text data, preprocessing typically involves tokenization (splitting text into individual words or phrases), stemming (reducing words to their root form, e.g., "running" to "run"), and removing stop words (common words like "the," "a," "is"). TF-IDF (Term Frequency-Inverse Document Frequency)

[3]See: `cv2.Sobel` in the OpenCV documentation.
[4]See: `keras.applications.VGG16` in the Keras documentation.

vectorization, implemented in scikit-learn's `TfidfVectorizer`[5], converts text into numerical vectors based on the frequency of words, weighting them by their importance across the entire dataset. Word embeddings, such as Word2Vec, GloVe, or FastText, represent words as dense vectors capturing semantic relationships. In the nostalgia example, TF-IDF might be used to identify keywords associated with past eras or events.

- **Audio Processing:** For audio data, feature extraction often involves calculating Mel-frequency cepstral coefficients (MFCCs), which represent the short-term power spectrum of a sound. Libraries like Librosa[6] provide functions for MFCC extraction. Spectrograms, visual representations of the frequencies present in a sound over time, can also be used as features. In the nostalgia example, MFCCs might be used to analyze the music or sound effects in the advertisements, potentially identifying characteristics of older music styles. For instance, in an example of coding nostalgia in video advertisements, visual features related to color palettes might be extracted (using functions like `cv2.calcHist` in OpenCV), the presence of vintage items detected (using pre-trained object detection models), and facial expressions analyzed (using facial recognition libraries). For audio, features like music genre and tempo might be extracted (using functions like `librosa.feature.mfcc` in Librosa). For text data from ad captions or voiceovers, NLP techniques would be used to identify nostalgic keywords or phrases. Feature extraction might also involve combining information from multiple data sources, such as merging structured data (e.g., product category) with features extracted from unstructured data (e.g., image features).

2. **Data Transformation:** This stage involves modifying *already numerical* features to improve model performance and ensure compatibility with AutoML algorithms. This is distinct from feature extraction, which creates numerical features from raw, unstructured data. Common

[5]See: `sklearn.feature_extraction.text.TfidfVectorizer` in the scikit-learn documentation.
[6]See: `librosa.feature.mfcc` in the Librosa documentation.

data transformation techniques include:

- **Log Transformation:** Applying a logarithm to skewed variables (variables where the distribution is not symmetrical) can make their distribution more normal. This is often beneficial for models that assume normality, such as linear regression. In the nostalgia example, if the distribution of "likes" on social media posts is highly skewed, a log transformation might be applied.
- **Standardization:** Scaling variables to have zero mean and unit variance is crucial for many machine learning algorithms, especially those sensitive to feature scaling, like neural networks. Standardization ensures that all features contribute equally to the model's learning process. This can be achieved using scikit-learn's `StandardScaler`[7].
- **One-Hot Encoding:** Categorical variables (e.g., product category, advertisement type) need to be converted into numerical representations. One-hot encoding creates a binary column for each category, indicating its presence or absence. This is commonly done using pandas' `get_dummies` function or scikit-learn's `OneHotEncoder`[8]. In the nostalgia example, if we have a categorical variable for "advertisement type" (e.g., "image," "video," "text"), one-hot encoding would create separate columns for each type.

3. **Data Splitting:** The *manually coded calibration subsample* (created in the previous section) must be randomly split into three distinct sets: training, validation, and test sets.

    - **Training Set:** This is the largest portion of the data and is used to train the AI models. The model learns the relationships between the input features and the target variable (e.g., nostalgia score) from this set.
    - **Validation Set:** This set is used to tune the hyperparameters of the AI models and to select the best-performing model *within* the AutoML framework. AutoML systems

[7] See: `sklearn.preprocessing.StandardScaler` in the scikit-learn documentation.
[8] See: `sklearn.preprocessing.OneHotEncoder` in the scikit-learn documentation.

use the validation set to evaluate different model architectures and configurations during their automated search process. This is a key distinction from standard machine learning, where the validation set is often used for manual hyperparameter tuning.

- **Test Set:** This set is used *only* to evaluate the final model's performance on *unseen* data. It is crucial that the test set remains completely separate and is *not* used during model training or hyperparameter tuning. This ensures an unbiased estimate of the model's generalization ability.

A common split is 70% for training, 15% for validation, and 15% for testing, but the optimal split may vary depending on the size of the dataset and the complexity of the model. Scikit-learn's `train_test_split` function[9] can be used to perform this split. It's important to perform this split *before* any data transformations to prevent data leakage, where information from the validation or test sets inadvertently influences the training process.

B.2.2 Training an AI Model to Code Nostalgia

This subsection demonstrates how to train an AI model to code the level of nostalgia evoked by text descriptions, using our running example. We cover two approaches: using Keras Tuner for more fine-grained control over the model architecture and using AutoKeras for a more automated approach. Both methods involve preprocessing the text data, building and training the model, and deploying the trained model to new data.

As described above, we assume we have a DataFrame (`calibration_df`) containing text descriptions of advertisements (in a column named `text`) and multiple manual coder ratings of nostalgia for each ad (in columns like `coder1`, `coder2`, `coder3`, etc.). Our goal is to train a model that can predict a single, continuous nostalgia score based on these text descriptions.

We first preprocess the text data, converting it into a numerical format suitable for machine learning. This involves creating a "bag-of-words" representation using `CountVectorizer`, where each unique word in the corpus becomes a feature, and the value of that feature represents

[9] See: `sklearn.model_selection.train_test_split` in the scikit-learn documentation.

the frequency of the word in a given text description. We need to resolve the multiple manual coder ratings into a single target variable. We do this by taking the mean of the individual coder ratings. Then, we build and train a neural network model using either Keras Tuner or AutoKeras. Finally, we deploy the trained model to predict nostalgia scores for new, unseen ad descriptions.

```
import autokeras as ak
import keras_tuner as kt
import numpy as np
import pandas as pd
import tensorflow as tf
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split
from tensorflow import keras
from tensorflow.keras import layers
import pickle

def preprocess_text_data(df, text_col='text', coder_cols=['coder1',
 ↪  'coder2', 'coder3'],
                         resolve_method='mean', test_size=0.2,
                          ↪  random_state=42, max_features=5000):
    """
    Preprocesses text data for AI model training (regression),
     ↪  including resolving multiple coder ratings.

    Args:
        df (pd.DataFrame): DataFrame with text data and coder ratings.
        text_col (str): Column with text data.
        coder_cols (list): Columns with individual coder ratings.
        resolve_method (str): Method to resolve ratings ('mean',
         ↪  'median').
        test_size (float): Proportion of data for validation (e.g., 0.2
         ↪  for 20%).
        random_state (int): Random seed for reproducibility.
        max_features (int): Max number of features (words) for
         ↪  CountVectorizer.

    Returns:
        tuple: (X_train, X_val, y_train, y_val, vectorizer,
         ↪  resolved_col_name).
               X_train, X_val: Numerical feature arrays.
               y_train, y_val: Resolved target variable arrays
                ↪  (continuous).
```

```
            vectorizer: Fitted CountVectorizer object.
            resolved_col_name: Name of the resolved target variable column.
    """

    # Check if coder_cols exist in the DataFrame
    if not all(col in df.columns for col in coder_cols):
        raise ValueError("One or more specified coder_cols do not exist in the DataFrame.")

    # 1. Resolve Multiple Coder Ratings into a single continuous target variable.
    resolved_col_name = 'resolved_nostalgia'  # Name of the new column for the resolved ratings
    if resolve_method == 'mean':
        df[resolved_col_name] = df[coder_cols].mean(axis=1)  # Calculate the mean of coder ratings.
    elif resolve_method == 'median':
        df[resolved_col_name] = df[coder_cols].median(axis=1)  # Calculate the median of coder ratings.
    else:
        raise ValueError("resolve_method must be 'mean' or 'median'.")

    # 2. Split data into training and validation sets.  No stratification needed for regression.
    train_df, val_df = train_test_split(df, test_size=test_size, random_state=random_state)
    if train_df.empty or val_df.empty:
        raise ValueError("One of the resulting dataframes (train_df or val_df) is empty.")

    # 3. Create a CountVectorizer to convert text to numerical vectors (bag-of-words).
    #    max_features limits the vocabulary size, which helps prevent overfitting and reduces memory usage.
    vectorizer = CountVectorizer(max_features=max_features, stop_words='english') # Exclude common English words.

    # 4. Fit the vectorizer on the training text data and transform both training and validation data.
    #    Fill missing text values with an empty string to avoid errors.
    X_train = vectorizer.fit_transform(train_df[text_col].fillna("")).toarray() # Convert sparse matrix to dense array.
```

```
    X_val = vectorizer.transform(val_df[text_col].fillna("")).toarray()

    # 5. Prepare the target variable (continuous nostalgia scores).
    y_train = train_df[resolved_col_name].values
    y_val = val_df[resolved_col_name].values

    return X_train, X_val, y_train, y_val, vectorizer,
    ↪ resolved_col_name  # Return all necessary data.


def model_builder(hp, input_shape):
    """
    Builds a Keras Sequential model (for regression) with
    ↪ hyperparameters tuned by Keras Tuner.

    Args:
        hp (kt.HyperParameters): Hyperparameter object provided by
        ↪ Keras Tuner.
        input_shape (tuple): Shape of the input data (number of
        ↪ features).

    Returns:
        keras.Sequential: A compiled Keras Sequential model.
    """
    model = keras.Sequential()
    model.add(layers.Input(shape=input_shape))  # Input layer,
    ↪ specifying the shape of the input data.

    # Tune the number of units in the first dense layer.
    # This is a key hyperparameter that controls the model's capacity.
    hp_units = hp.Int('units', min_value=32, max_value=512, step=32)
    model.add(layers.Dense(units=hp_units, activation='relu'))  # Dense
    ↪ layer with ReLU activation.  ReLU is a common choice for hidden
    ↪ layers.

    # Optionally add Batch Normalization (helps with training stability
    ↪ and can allow for higher learning rates).
    if hp.Boolean('use_batchnorm', default=False):
        model.add(layers.BatchNormalization())

    # Tune the dropout rate (helps prevent overfitting by randomly
    ↪ dropping out units during training).
    hp_dropout = hp.Float('dropout', min_value=0.0, max_value=0.5,
    ↪ step=0.1)
```

```
    model.add(layers.Dropout(rate=hp_dropout))  # Dropout layer.

    # Output layer: 1 unit, linear activation for regression (predicting a continuous value).
    model.add(layers.Dense(1, activation='linear'))

    # Tune the learning rate for the Adam optimizer.
    hp_learning_rate = hp.Choice('learning_rate', values=[1e-2, 1e-3, 1e-4])

    # Compile the model.  Use Mean Squared Error (MSE) loss for regression.
    model.compile(optimizer=keras.optimizers.Adam(learning_rate=hp_learning_rate),
                  loss='mean_squared_error',  # MSE loss function
                  metrics=['mae'])  # Mean Absolute Error (MAE) as an evaluation metric.

    return model

def tune_hyperparameters(X_train, y_train, X_val, y_val, directory='my_dir', project_name='nostalgia_regression', max_epochs=10, factor=3):
    """
    Tunes hyperparameters using Keras Tuner (for regression).

    Args:
        X_train (np.array): Training data features.
        y_train (np.array): Training data labels (continuous).
        X_val (np.array): Validation data features.
        y_val (np.array): Validation data labels (continuous).
        directory (str): Directory to store tuner results.
        project_name (str): Project name for the tuner.
        max_epochs (int): Max epochs to train each model.
        factor (int): Reduction factor for Hyperband tuner.

    Returns:
        kt.HyperParameters: Optimal hyperparameters found by the tuner.
    """
    # Instantiate the Hyperband tuner.  Minimize validation MAE.
    # Hyperband is an efficient algorithm for hyperparameter optimization that focuses on exploring a wide range of
    # hyperparameters and quickly discarding poorly performing configurations.
```

```
    # The 'factor' parameter controls the aggressiveness of the early-stopping.
    tuner = kt.Hyperband(lambda hp: model_builder(hp, input_shape=(X_train.shape[1],)),
                         objective=kt.Objective("val_mae", direction="min"),
                         max_epochs=max_epochs,
                         factor=factor,
                         directory=directory,
                         project_name=project_name,
                         overwrite=True)  # Overwrite previous results if they exist.

    # Add a callback to stop training early if validation loss doesn't improve.
    # This prevents overfitting and saves computation time.
    stop_early = keras.callbacks.EarlyStopping(monitor='val_loss', patience=5)

    # Perform the hyperparameter search.
    try:
        tuner.search(X_train, y_train, epochs=max_epochs, validation_data=(X_val, y_val), callbacks=[stop_early])
    except Exception as e:
        print(f"An error occurred during hyperparameter search: {e}")
        return None  # Return None if an error occurs.

    # Get the optimal hyperparameters.
    best_hps = tuner.get_best_hyperparameters(num_trials=1)[0]
    print(f"""
    The hyperparameter search is complete. The optimal number of units in the first densely-connected
    layer is {best_hps.get('units')}, the optimal dropout rate is {best_hps.get('dropout')}, and the optimal learning rate for the optimizer
    is {best_hps.get('learning_rate')}.
    """)
    return best_hps


def train_and_evaluate_model(tuner, best_hps, X_train, y_train, X_val, y_val, epochs=50):
    """
    Builds, trains, and evaluates a Keras model (for regression).
```

```
    Args:
        tuner (kt.Hyperband): Keras Tuner object.
        best_hps (kt.HyperParameters): Optimal hyperparameters.
        X_train (np.array): Training data features.
        y_train (np.array): Training data labels (continuous).
        X_val (np.array): Validation data features.
        y_val (np.array): Validation data labels (continuous).
        epochs (int): Number of epochs to train.

    Returns:
        tuple: Trained model, evaluation results (loss, mae), and
        ↪ training history.
    """
    # Build the model with the optimal hyperparameters.
    model = tuner.hypermodel.build(best_hps)

    # Train the model.
    history = model.fit(X_train, y_train, epochs=epochs,
    ↪ validation_data=(X_val, y_val))

    # Find the best epoch based on validation MAE (we want to
    ↪ *minimize* MAE).
    val_mae_per_epoch = history.history['val_mae']
    best_epoch = val_mae_per_epoch.index(min(val_mae_per_epoch)) + 1  #
    ↪ Find epoch with *minimum* val_mae
    print('Best epoch: %d' % (best_epoch,))

    # Rebuild the model (to ensure a clean state).
    hypermodel = tuner.hypermodel.build(best_hps)

    # Retrain the model using the best number of epochs.  This is
    ↪ crucial for optimal performance.
    hypermodel.fit(X_train, y_train, epochs=best_epoch,
    ↪ validation_data=(X_val, y_val))

    # Evaluate the model on the validation set.
    eval_result = hypermodel.evaluate(X_val, y_val)
    print("[Validation loss, Validation MAE]:", eval_result)

    # Save the trained model.
    hypermodel.save("nostalgia_model.h5")
    return hypermodel, eval_result, history  # Return the training
    ↪ history
```

```python

def train_autokeras_model(df, text_col='text', coder_cols=['coder1', 'coder2', 'coder3'],
                          resolve_method='mean', max_trials=10, epochs=20,
                          project_name="nostalgia_autokeras"):
    """
    Trains a text regression model using AutoKeras, including resolving multiple coder ratings.

    Args:
        df (pd.DataFrame): DataFrame with training data.
        text_col (str): Column with text data.
        coder_cols (list): List of columns with individual coder ratings.
        resolve_method (str): Method to resolve ratings ('mean', 'median').
        max_trials (int): Maximum number of different models to try.
        epochs (int): Number of epochs to train each model.
        project_name (str): Name of the project.

    Returns:
        tuple: Trained AutoKeras model and evaluation results.
    """

    # 1. Resolve Multiple Coder Ratings (same as in preprocess_text_data)
    resolved_col_name = 'resolved_nostalgia'
    if resolve_method == 'mean':
        df[resolved_col_name] = df[coder_cols].mean(axis=1) # Calculate the mean.
    elif resolve_method == 'median':
        df[resolved_col_name] = df[coder_cols].median(axis=1) # Calculate the median
    else:
        raise ValueError("resolve_method must be 'mean' or 'median'.")

    # 2. Split data (no stratification for regression)
    train_df, val_df = train_test_split(df, test_size=0.2, random_state=42)
    if train_df.empty or val_df.empty:
        raise ValueError("One of the resulting dataframes (train_df or val_df) is empty.")

```

```
    x_train = train_df[text_col].fillna("").values  # Prepare text data (handle missing values).
    y_train = train_df[resolved_col_name].values  # Target variable.
    x_val = val_df[text_col].fillna("").values  # Prepare text data (handle missing values).
    y_val = val_df[resolved_col_name].values  # Target variable.

    # 3. Initialize and train the text regressor.
    #    AutoKeras automatically handles feature preprocessing and model selection.
    #    'max_trials' controls the number of different model architectures to try.
    clf = ak.TextRegressor(overwrite=True, max_trials=max_trials, project_name=project_name)  # Use TextRegressor.
    try:
        clf.fit(x_train, y_train, epochs=epochs, validation_data=(x_val, y_val))
    except Exception as e:
        print(f"An error occurred during AutoKeras training: {e}")
        return None, None  # Return None if an error occurs.

    # 4. Evaluate and save the model.
    results = clf.evaluate(x_val, y_val)  # Returns loss and metrics (e.g., MAE).
    print(f"Evaluation Results: {results}")
    model = clf.export_model()
    model.save("nostalgia_autokeras_model")  # Save in the TensorFlow SavedModel format.
    return model, results

def deploy_ai_model(model, vectorizer, new_data, text_col='text', id_col='Ad ID', prediction_col='nostalgia_ai'):
    """
    Deploys the trained AI model (regression) to code new data. Handles Keras and AutoKeras models.

    Args:
        model (keras.Model or autokeras.Regressor): Trained Keras or AutoKeras model.
        vectorizer (CountVectorizer): CountVectorizer used during training.  *MUST* be the same one.
        new_data (pd.DataFrame): DataFrame with new data to be coded.
        text_col (str): Column with text data.
        id_col (str): Column with unique identifier (e.g., 'Ad ID').
```

```
        prediction_col (str): Column to store predictions (continuous values).

    Returns:
        pd.DataFrame: Copy of DataFrame with AI-coded predictions (continuous).
    """
    # Check if the vectorizer has been fitted (crucial for consistency).
    if not hasattr(vectorizer, 'vocabulary_'):
        raise ValueError("The provided CountVectorizer has not been fitted. Please fit it on the training data first.")

    # Verify that the new data has the expected text column.
    if text_col not in new_data.columns:
        raise ValueError(f"The new_data DataFrame does not contain the specified text column: {text_col}")

    # Preprocess the new data using the *same* vectorizer used during training.
    X_new = vectorizer.transform(new_data[text_col].fillna("")).toarray()

    # Check that the number of features in X_new matches the vectorizer's vocabulary size.
    if X_new.shape[1] != len(vectorizer.vocabulary_):
        raise ValueError(f"The number of features in the new data ({X_new.shape[1]}) does not match the vectorizer's vocabulary size ({len(vectorizer.vocabulary_)}).  Ensure you are using the same preprocessing steps and the same fitted vectorizer.")

    # Handle different model types and output shapes.
    if isinstance(model, keras.Model):
        # Keras model: .predict() should return continuous values directly.
        predictions = model.predict(X_new).flatten()  # Get continuous predictions.
    elif hasattr(model, 'predict'):  # Likely an AutoKeras model.
        predictions = model.predict(X_new)
        if predictions.ndim > 1 and predictions.shape[-1] > 1:  # Multiclass (shouldn't happen in regression).
            predictions = np.argmax(predictions, axis=-1) # Handle as a precaution.
```

```
        elif predictions.ndim > 1:  # (n_samples, 1) format.
            predictions = predictions.flatten()  # Flatten to a 1D array.
    else:
        raise TypeError("Unsupported model type. Must be a Keras or AutoKeras model.")

    # Add the continuous predictions to a *copy* of the DataFrame.
    new_data = new_data.copy() # Create a copy to avoid modifying the original DataFrame.
    new_data[prediction_col] = predictions  # Store continuous predictions.
    return new_data

# --- Example Usage ---
# 1. Prepare Data (Assuming you have a DataFrame 'calibration_df')
#    with 'text', 'coder1', 'coder2', 'coder3' columns)
# calibration_df is the manually-coded calibration subsample
X_train, X_val, y_train, y_val, vectorizer, resolved_col_name = preprocess_text_data(calibration_df)

# Save the vectorizer
with open('vectorizer.pkl', 'wb') as f:
    pickle.dump(vectorizer, f)

# Load the vectorizer later
with open('vectorizer.pkl', 'rb') as f:
    loaded_vectorizer = pickle.load(f)

# 2. Keras Tuner (Option 1)
best_hps = tune_hyperparameters(X_train, y_train, X_val, y_val)
# hypermodel is the trained Keras model
hypermodel, eval_result, history = train_and_evaluate_model(tuner, best_hps, X_train, y_train, X_val, y_val)

# 3. AutoKeras (Option 2)
# model, results = train_autokeras_model(calibration_df)
# To load the saved AutoKeras model later:
# loaded_model is the trained and loaded AutoKeras model
# loaded_model = tf.keras.models.load_model("nostalgia_autokeras_model", custom_objects=ak.CUSTOM_OBJECTS)

# 4. Deploy the model (using Keras model as an example)
```

```
#    Assume 'full_df' contains the full dataset with a 'text' column and an 'Ad ID' column.
# vectorizer is the fitted CountVectorizer from the preprocessing step
full_df = deploy_ai_model(hypermodel, vectorizer, full_df) # Use 'loaded_model' for AutoKeras

print(full_df.head())

```

B.3 Estimating Moment Matrices in the Calibration Subsample

After developing and deploying AI models to code the marketing constructs of interest (as described in the previous subsection), the next crucial step is to estimate the moment matrices using the calibration subsample. These matrices are essential for correcting the biases introduced by AI-coding errors. Specifically, our proposed estimators (detailed in later sections) adjust the canonical estimators (e.g., 2SLS, 3SLS) by incorporating information about the coding errors, which is captured within these moment matrices.

This section provides a detailed guide to estimating the moment matrices in the calibration subsample. We organize the subsection into sub-sections, the former presenting details for single-equation models (2SLS) and the latter for system-of-equations models (3SLS). In each case, we define the matrices to be calculated, then provide step-by-step instructions for computing these matrices, and finally provide code to compute these matrices.

In addition, we discuss how to assess the identification conditions. The identification conditions formalize the following tradeoffs: The accuracy of the estimated moment matrices depends on the size of the calibration subsample. Larger subsamples generally lead to more precise estimates but also increase the cost of manual coding. Furthermore, more complex models are better able to account for additional factors (e.g., they can enhance statistical control through the inclusion of more control variables). However, more complex models may increase the identification burden whereby the instruments need to be sufficiently informative of the endogenous variables, and the calibration subsample needs to be long enough and precisely coded enough, to ensure robust

estimation.

B.3.1 Single-Equation Models (2SLS)

B.3.1.1 Defining the Moment Matrices (2SLS) To implement the bias-corrected 2SLS estimator, we need to estimate the following moment matrices using the *calibration subsample*:

1. $\hat{s}_{\xi\zeta,2SLS} = \frac{1}{N_s}\sum_{i=1}^{N_s}\hat{\xi}_i\hat{\zeta}_i'$: The sample cross-product matrix of the *observed* regressors and instruments. This matrix captures the relationships between the observed variables (both AI-coded and directly observed) in the model.

2. $\hat{s}_{\zeta\zeta,2SLS} = \frac{1}{N_s}\sum_{i=1}^{N_s}\hat{\zeta}_i\hat{\zeta}_i'$: The sample cross-product matrix of the *observed* instruments. This matrix captures the relationships among the instruments themselves.

3. $\hat{\psi}_{2SLS} = \frac{1}{N_s}\sum_{i=1}^{N_s}\hat{\zeta}_i\eta_{\xi i}'$: The sample cross-product matrix of the *observed* instruments and the coding errors. This matrix is *crucial* for bias correction, as it captures the systematic relationship between the instruments and the coding errors.

Where:

- N_s is the number of observations in the calibration subsample.
- $\hat{\xi}_i$ represents the observed values of the regressors for observation i in the calibration subsample. This includes *both* AI-coded regressors and any observed (non-AI-coded) regressors.
- $\hat{\zeta}_i$ represents the observed values of the instruments for observation i in the calibration subsample.
- $\eta_{\xi i}$ represents the coding error for observation i. This is the *difference* between the manually coded ("true") value and the AI-coded value for each AI-coded regressor.

B.3.1.2 Estimating the Moment Matrices (2SLS) The estimation process involves these steps:

1. **Compute AI-Coding Errors:** For each observation in the calibration subsample, and for each AI-coded variable, compute the coding error ($\eta_{\xi i}$) as the difference between the manually coded value and the AI-coded value.

2. **Construct Matrices:** Create the matrices $\hat{\xi}$, $\hat{\zeta}$, and η_{ξ} using the observed values and the computed coding errors.

3. **Calculate Cross-Products:** Compute the sample analogs of the moment matrices ($\hat{s}_{\xi\zeta,2SLS}$, $\hat{s}_{\zeta\zeta,2SLS}$, and $\hat{\psi}_{2SLS}$) using the formulas defined above.

```python
import numpy as np
import pandas as pd

def calculate_moment_matrices_2sls(calibration_df, ai_coded_regressors,
                                   ai_coded_instruments,
                                    ↪ manual_regressors,
                                   observed_regressors=[]):
    """
    Calculates the moment matrices required for 2SLS bias correction.

    Args:
        calibration_df (pd.DataFrame): DataFrame containing the
         ↪ calibration subsample data.
                                       Must include manually coded
                                        ↪ values, AI-coded values, and
                                        ↪ instruments.
        ai_coded_regressors (list): List of column names representing
         ↪ AI-coded regressors.
        ai_coded_instruments (list): List of column names representing
         ↪ AI-coded instruments.
        manual_regressors (list): List of column names representing the
         ↪ manually coded ("true") values
                                  corresponding to the
                                   ↪ ai_coded_regressors.
        observed_regressors (list, optional): List of column names for
         ↪ observed regressors (not AI-coded).
                                              Defaults to an empty list.

    Returns:
```

```
        tuple: Estimated moment matrices (s_xizeta_2sls,
        ↪ s_zetazeta_2sls, psi_hat_2sls).
            s_xizeta_2sls: Cross-product of observed regressors and
            ↪ instruments.
            s_zetazeta_2sls: Cross-product of observed instruments.
            psi_hat_2sls: Cross-product of observed instruments and
            ↪ coding errors.
    """

    # 1. Compute the AI-coding errors.
    for ai_col, manual_col in zip(ai_coded_regressors,
    ↪ manual_regressors):
        calibration_df[f'eta_{ai_col}'] = calibration_df[manual_col] -
        ↪ calibration_df[ai_col]

    # 2. Construct the matrices.
    xi_hat = calibration_df[ai_coded_regressors +
    ↪ observed_regressors].values
    zeta_hat = calibration_df[ai_coded_instruments].values
    eta_cols = [f'eta_{col}' for col in ai_coded_regressors]
    eta_xi = calibration_df[eta_cols].values  # Matrix of coding errors

    # 3. Compute the moment matrices.
    N_s = calibration_df.shape[0]
    s_xizeta_2sls = (xi_hat.T @ zeta_hat) / N_s
    s_zetazeta_2sls = (zeta_hat.T @ zeta_hat) / N_s
    psi_hat_2sls = (zeta_hat.T @ eta_xi) / N_s

    return s_xizeta_2sls, s_zetazeta_2sls, psi_hat_2sls
```

B.3.1.3 Code for Estimating the Moment Matrices (2SLS)

```
import numpy as np

def check_identification_conditions_2sls(s_xizeta_2sls,
↪ s_zetazeta_2sls, psi_hat_2sls):
    """
    Checks the identification conditions for the bias-corrected 2SLS
    ↪ estimator.

    Args:
```

```
        s_xizeta_2sls (np.ndarray):  Cross-product matrix of observed
        ↪  regressors and instruments.
        s_zetazeta_2sls (np.ndarray): Cross-product matrix of observed
        ↪  instruments.
        psi_hat_2sls (np.ndarray):   Cross-product matrix of observed
        ↪  instruments and coding errors.

    Returns:
        dict: Condition numbers of relevant matrices (s_xizeta_2sls,
        ↪  s_zetazeta_2sls, and 'a_2sls').
              Larger condition numbers (e.g., > 30) suggest potential
              ↪  identification issues.
    """
    # Compute condition numbers.
    cond_s_xizeta_2sls = np.linalg.cond(s_xizeta_2sls)
    cond_s_zetazeta_2sls = np.linalg.cond(s_zetazeta_2sls)

    # Construct the matrix 'a_2sls' (BiasFactor).
    a_2sls = np.eye(s_xizeta_2sls.shape[0]) -
    ↪  np.linalg.inv(s_xizeta_2sls @ np.linalg.inv(s_zetazeta_2sls) @
    ↪  s_xizeta_2sls.T) @ s_xizeta_2sls @
    ↪  np.linalg.inv(s_zetazeta_2sls) @ psi_hat_2sls
    cond_a_2sls = np.linalg.cond(a_2sls)

    return {
        "cond_s_xizeta_2sls": cond_s_xizeta_2sls,
        "cond_s_zetazeta_2sls": cond_s_zetazeta_2sls,
        "cond_a_2sls": cond_a_2sls
    }
```

B.3.1.4 Code for Checking Identification (2SLS)

B.3.1.5 Identification (2SLS) Before applying the bias-corrected 2SLS estimator, it's crucial to assess whether the model is *identified*. Identification, in this context, means that we have enough information in our data and our model structure to obtain unique and reliable estimates of the parameters. In the standard 2SLS setting, identification is often assumed if the basic order condition (the number of instruments is greater than or equal to the number of endogenous regressors) is met. However, the presence of AI-coded variables, with their potential for correlated errors, introduces additional complexities that can compromise identification *even if* the standard

conditions hold.

Specifically, for our bias-corrected 2SLS estimator to be well-defined and provide meaningful results, we need to ensure two main conditions:

1. **Sufficient Instrument Strength and Independence:** This condition is analogous to the standard requirement in 2SLS, and related to the rank of instruments (and regressors). The matrices $\hat{s}_{\xi\zeta,2SLS}$ (the cross-product of the *observed* regressors and instruments) and $\hat{s}_{\zeta\zeta,2SLS}$ (the cross-product of the *observed* instruments) must both have full column rank. Practically, this means two things. First, the instruments must be sufficiently correlated with the endogenous regressors. If this correlation is weak, the estimates become highly unstable. Second, the instruments themselves must be linearly independent. If there's perfect multicollinearity among the instruments, we can't separately identify their effects. In traditional settings without AI-coded data, this is primarily ensured during the design stage by selecting suitable instruments, and checking for it computationally involves examining the moments of observed data. In our model setup, the presence of $\hat{s}_{\xi\zeta,2SLS}$ and $\hat{s}_{\zeta\zeta,2SLS}$ highlights how the *coding errors* in either regressors or instruments are essential, and that coding errors, regressors, and instruments must be *jointly* considered when testing whether the data contain sufficient signal.

2. **Separability of AI-Coding Error Effects:** The matrix $(I_g - \widehat{\text{BiasFactor}}_{2SLS})$, which is central to our bias correction, must be invertible. This condition is unique to our framework, addressing AI-coding error. Intuitively, this means that the *effect* of the AI-coding error must be statistically distinguishable from the *effect* of the true, underlying variable. If the AI-coding error is *too* closely related to the true variable (and other variables in the model), it becomes impossible to separate their influences, making bias correction ineffective. Mathematically, this invertibility condition is equivalent to requiring that the true regressors and true instruments are sufficiently correlated, even *after* accounting for the impact of the AI-coding error. As we outline later in Equation (47), a sufficient condition requires

a non-zero probability limit of sample moments of true variables, $s_{\xi\zeta}$, and coding errors (ψ_{2SLS}).

B.3.1.6 Assessing Identification Using Condition Numbers (2SLS) The practical way to check these identification conditions is to examine the *condition numbers* of the relevant matrices: $\hat{s}_{\xi\zeta,2SLS}$, $\hat{s}_{\zeta\zeta,2SLS}$, and $(I_g - \widehat{\text{BiasFactor}}_{2SLS})$. The condition number of a matrix is, intuitively, a measure of how "close" that matrix is to being singular (non-invertible). It's calculated as the ratio of the matrix's largest singular value to its smallest singular value.

- **Large Condition Number:** A very large condition number indicates that the matrix is nearly singular. This means that small changes in the data could lead to large changes in the matrix inverse, making the estimates highly sensitive and unreliable. This implies practical problems for inference, estimation, and therefore the *design* of the marketing data collection.
- **Small Condition Number:** A relatively small condition number suggests that the matrix is well-conditioned, and its inverse is stable.

The `check_identification_conditions_2sls` function (provided in the code examples) computes these condition numbers. As input, the function requires the estimated moment matrices, and as output, it returns the condition numbers. These outputs can be interpreted as follows:

- $\hat{s}_{\xi\zeta,2SLS}$ **and** $\hat{s}_{\zeta\zeta,2SLS}$**:** Large condition numbers here typically indicate *weak instruments* (if the correlation between instruments and endogenous regressors is low) or *multicollinearity* (if the regressors or instruments are too highly correlated with each other).

- $(I_g - \widehat{\textbf{BiasFactor}}_{2SLS})$**:** A large condition number for this matrix is a more specific signal of trouble related to the AI coding. It suggests that the AI-coding errors are not statistically distinguishable enough from the AI-coded variables, hindering the bias correction. This problem is more challenging to solve because some overlap might be expected by construction, given the assumption of a calibration dataset.

B.3.1.7 Remedial Measures A commonly used rule of thumb is that condition numbers greater than 30 are cause for concern. However, this is *not* a strict cutoff. The acceptable threshold depends on the specific context, the size of the dataset, and the sensitivity of your results to small perturbations in the data.

If the identification checks reveal large condition numbers, indicating potential problems, researchers have several options to consider, ordered by increasing degree of modification to the overall study (and not just the estimation method):

1. **Examine the Data and Model:** Before making major changes, carefully re-examine the data and the model specification. Are there any obvious errors or omissions? Are all variables properly defined and measured? Sometimes, a simple correction can resolve the issue.

2. **Increase the Subsample Data Collection:** For small datasets, there may not be sufficient information to support an instrumental variable estimate (e.g., if some exogenous conditions are not measured). Our framework accommodates adding information by increasing N_s, i.e., by measuring a larger set of true outcomes in a calibration sample. This also implies that researchers should *design* their study to allow for the collection of a calibration subsample.

3. **Instrument Strength:** Consider revisiting the *choice* of instruments. Are they likely to influence the endogenous variable(s) but not directly related to the error term in the structural model? Are they sufficiently powerful?

4. **Model Complexity:** If a more parsimonious or theoretically supported model would suffice, removing an element with weaker correlation could aid inference. However, such cases must be supported conceptually; it may also reflect an underpowered calibration set.

5. **Model and Hypotheses:** The nature of inference might not be supported by available data. In this case, it may be useful to reflect whether identification, under the structure imposed by an assumed model and model parameters, supports tests of focal hypotheses.

B.3.2 System-of-Equations Models (3SLS)

B.3.2.1 Defining the Moment Matrices (3SLS) For a system of G equations, we need to calculate moment matrices for *each* equation $g = 1, \ldots, G$. We assume the instruments are the *same* across all equations, which simplifies the notation and is common in many marketing applications. If the instruments differ across equations, the formulas would need to be adjusted accordingly (as shown in Web Appendix D).

For each equation g, we need:

1. $\hat{s}_{\xi_g\zeta,3SLS} = \frac{1}{N_s}\sum_{i=1}^{N_s} \hat{\xi}_{gi}\hat{\zeta}_i'$: The sample cross-product matrix of the *observed* regressors for equation g and the instruments.

2. $\hat{s}_{\zeta\zeta,3SLS} = \frac{1}{N_s}\sum_{i=1}^{N_s} \hat{\zeta}_i\hat{\zeta}_i'$: The sample cross-product matrix of the *observed* instruments. This is the *same* for all equations if the instruments are common.

3. $\hat{\psi}_{g,3SLS} = \frac{1}{N_s}\sum_{i=1}^{N_s} \hat{\zeta}_i\eta_{\xi gi}'$: The sample cross-product matrix of the *observed* instruments and the coding errors for equation g.

Where:

- N_s is the number of observations in the calibration subsample.
- $\hat{\xi}_{gi}$ represents the observed values of the regressors for observation i in equation g in the calibration subsample.
- $\hat{\zeta}_i$ represents the observed values of the instruments for observation i in the calibration subsample (same for all equations).
- $\eta_{\xi gi}$ represents the coding error for observation i in equation g.

B.3.2.2 Estimating the Moment Matrices (3SLS) The estimation process for 3SLS is analogous to 2SLS, but it's performed for each equation:

1. **Compute AI-Coding Errors:** For each observation, equation, and AI-coded variable, compute the coding error.

2. **Construct Matrices:** Create the matrices $\hat{\xi}_g$, $\hat{\zeta}$, and $\eta_{\xi g}$ for each equation.

3. **Calculate Cross-Products:** Compute the sample analogs of the moment matrices ($\hat{s}_{\xi_g \zeta, 3SLS}$, $\hat{s}_{\zeta\zeta, 3SLS}$, and $\hat{\psi}_{g,3SLS}$) for each equation.

```python
import numpy as np
import pandas as pd

def calculate_moment_matrices_3sls(calibration_df, equations_data, instruments):
    """
    Calculates the moment matrices required for 3SLS bias correction.

    Args:
        calibration_df (pd.DataFrame): DataFrame containing the calibration subsample data.
        equations_data (list): List of dictionaries, one for each equation.
                               Each dictionary should contain:
                                 'ai_coded_regressors': List of AI-coded regressor column names.
                                 'manual_regressors': List of corresponding manual regressor column names.
                                 'observed_regressors': List of observed regressor column names.
        instruments (list): List of column names representing the instruments (common to all equations).

    Returns:
        tuple: (s_xizeta_matrices, s_zetazeta_3sls, psi_hat_matrices)
               s_xizeta_matrices: List of cross-product matrices (observed regressors and instruments) for each equation.
               s_zetazeta_3sls: Cross-product matrix of observed instruments (common to all equations).
               psi_hat_matrices: List of cross-product matrices (observed instruments and coding errors) for each equation.
    """

```

```
    s_xizeta_matrices = []
    psi_hat_matrices = []
    N_s = calibration_df.shape[0]
    zeta_hat = calibration_df[instruments].values
    s_zetazeta_3sls = (zeta_hat.T @ zeta_hat) / N_s # Same for all
    ↪  equations

    for eq_data in equations_data:
        ai_coded = eq_data['ai_coded_regressors']
        manual = eq_data['manual_regressors']
        observed = eq_data['observed_regressors']

        # Compute coding errors
        for ai_col, manual_col in zip(ai_coded, manual):
            calibration_df[f'eta_{ai_col}'] =
            ↪  calibration_df[manual_col] - calibration_df[ai_col]

        # Construct matrices for this equation
        xi_hat = calibration_df[ai_coded + observed].values
        eta_cols = [f'eta_{col}' for col in ai_coded]
        eta_xi = calibration_df[eta_cols].values

        # Calculate moment matrices for this equation
        s_xizeta_eq = (xi_hat.T @ zeta_hat) / N_s  # Equation-specific
        psi_hat_eq = (zeta_hat.T @ eta_xi) / N_s    # Equation-specific

        s_xizeta_matrices.append(s_xizeta_eq)
        psi_hat_matrices.append(psi_hat_eq)

    return s_xizeta_matrices, s_zetazeta_3sls, psi_hat_matrices
```

B.3.2.3 Code for Estimating the Moment Matrices (3SLS)

B.3.2.4 Identification (3SLS) Identification in system-of-equations models should be assessed at the *equation level*. This means that the identification conditions for 2SLS (discussed above) apply to *each equation* within the system. We use the `check_identification_conditions_2sls` function on the moment matrices calculated for *each equation individually* to check for identification issues. If any single equation fails the identification checks, the entire 3SLS system is not identified. The 3SLS estimator relies on the first-stage (2SLS) estimates, so if any of those

first-stage estimates are not identified, the entire system is compromised. Therefore, there is no separate identification check function for 3SLS. We reuse the 2SLS identification check, applied equation-by-equation. Below is an example of such a use case:

```
# --- Preparing Data (Illustrative) ---
# In a real application, 'calibration_df' would be your
# manually coded calibration subsample.
data = {
    'Ad ID': range(1, 101),
    'text': [f'Ad text {i}' for i in range(1, 101)],
    'z_nost_manual': np.random.randint(1, 8, size=100),  # Example: Manually coded nostalgia
    'z_nost_ai': np.random.rand(100) * 7,              # Example: AI-coded nostalgia
    'x_fea': np.random.rand(100),                     # Example: Observed regressor
    'w_her_ai': np.random.rand(100),                   # Example: AI-coded instrument 1
    'w_cost_ai': np.random.rand(100),                  # Example: AI-coded instrument 2
    'z_price_manual': np.random.rand(100) * 10,        # Example: Manually coded price
    'z_price_ai': np.random.rand(100) * 10             # Example: AI-coded price
}
calibration_df = pd.DataFrame(data)

# --- 2SLS Moment Matrices and Identification Check ---
ai_coded_regressors_2sls = ['z_nost_ai']
ai_coded_instruments_2sls = ['w_her_ai', 'w_cost_ai']
manual_regressors_2sls = ['z_nost_manual']
observed_regressors_2sls = ['x_fea']

s_xizeta_2sls, s_zetazeta_2sls, psi_hat_2sls = calculate_moment_matrices_2sls(
    calibration_df,
    ai_coded_regressors_2sls,
    ai_coded_instruments_2sls,
    manual_regressors_2sls,
    observed_regressors_2sls
)

condition_numbers_2sls = check_identification_conditions_2sls(s_xizeta_2sls, s_zetazeta_2sls, psi_hat_2sls)
```

```python
print("2SLS Condition Numbers:", condition_numbers_2sls)

# --- 3SLS Moment Matrices ---
# Define equations data (example with two equations)
equations_data = [
    {
        'ai_coded_regressors': ['z_nost_ai'],
        'manual_regressors': ['z_nost_manual'],
        'observed_regressors': ['x_fea']
    },
    {
        'ai_coded_regressors': ['z_price_ai'],
        'manual_regressors': ['z_price_manual'],
        'observed_regressors': []  # No observed regressors in this example equation
    }
]

# Common instruments for both equations
instruments = ['w_her_ai', 'w_cost_ai']

# Calculate 3SLS moment matrices
s_xizeta_matrices, s_zetazeta_3sls, psi_hat_matrices = calculate_moment_matrices_3sls(
    calibration_df, equations_data, instruments
)

print("\n3SLS Moment Matrices:")
for i, (s_xizeta, psi_hat) in enumerate(zip(s_xizeta_matrices, psi_hat_matrices)):
    print(f"Equation {i+1}:")
    print("s_xizeta:", s_xizeta)
    print("psi_hat:", psi_hat)
    # Check identification for EACH equation using the 2SLS check
    condition_numbers = check_identification_conditions_2sls(s_xizeta, s_zetazeta_3sls, psi_hat) # s_zetazeta is the same
    print(f"  Condition Numbers (Equation {i+1}):", condition_numbers)

print("s_zetazeta (common to all equations):", s_zetazeta_3sls)
```

B.4 Estimating the Econometric Models

Once the moment matrices have been estimated using the calibration subsample, the next step is to incorporate them into the econometric models to obtain bias-corrected estimates of the parameters of interest. This section provides a practical guide to implementing the proposed estimators for both single-equation models (e.g., OLS, 2SLS) and system-of-equations models (e.g., 3SLS). We present the general form of the estimators, discuss the necessary identification conditions, and provide algorithms and code for their implementation. We also address the computation of standard errors using bootstrap methods and discuss model diagnostics and validation. It is assumed the model is correctly specified and the usual assumptions for consistency of the canonical estimators hold (e.g., for 2SLS, that the instruments are relevant and valid in the absence of coding error).

B.4.1 Single-Equation Models (OLS, 2SLS)

In the single-equation context, we consider models of the form:

$$y = \xi\delta + \varepsilon, \tag{11}$$

where y is the dependent variable, $\xi = [z\ x]$ is the matrix of regressors (including both AI-coded variables z and observed variables x), δ is the vector of coefficients, and ε is the error term. When some of the regressors are endogenous, we also have a matrix of instruments ζ.

The proposed bias-corrected 2SLS estimator, $d_{\text{2SLS}}^{\text{con}}$, is given by the solution to the following equation:

$$\left[I_g - \left(\hat{s}_{\xi\zeta} \hat{s}_{\zeta\zeta}^{-1} \hat{s}_{\xi\zeta}' \right)^{-1} \hat{s}_{\xi\zeta} \hat{s}_{\zeta\zeta}^{-1} \hat{\psi}_{2SLS} \right] d_{\text{2SLS}}^{\text{con}} = d_{\text{2SLS}}, \tag{12}$$

where d_{2SLS} is the canonical 2SLS estimator, $\hat{s}_{\xi\zeta}$ and $\hat{s}_{\zeta\zeta}$ are the estimated cross-product matrices of the regressors and instruments, $\hat{\psi}_{2SLS}$ is the estimated cross-product matrix of the instruments and coding errors, and I_g is the identity matrix of dimension g (the number of regressors).

We use Algorithm 1 from Web Appendix C, now with updated notation:

Algorithm 4 Algorithm for Inference in Single Equation Marketing Models

for $b \leftarrow 1, nB$ **do**

Draw bootstrap sample Bootstrap Calibration Subsample$[b]$ of size N_s from Data$_s$ with replacement.

Infer $\hat{\psi}_{2SLS} = \frac{\hat{\zeta}'\eta_{\xi}}{N_s}$ in Bootstrap Calibration Subsample$[b]$.

Draw bootstrap sample Bootstrap Full Sample$[b]$ of size N_f from Data$_f$ with replacement.

Compute the canonical 2SLS estimator, $d_{2\text{SLS}}[b] = \left[\hat{\xi}'\hat{\zeta}\left(\hat{\zeta}'\hat{\zeta}\right)^{-1}\hat{\zeta}'\hat{\xi}\right]^{-1}\hat{\xi}'\hat{\zeta}\left(\hat{\zeta}'\hat{\zeta}\right)^{-1}\hat{\zeta}'y$, in Bootstrap Full Sample$[b]$.

Infer $\hat{s}_{\xi\zeta} = \frac{\hat{\xi}'\hat{\zeta}}{N_f}$ and $\hat{s}_{\zeta\zeta} = \frac{\hat{\zeta}'\hat{\zeta}}{N_f}$ in Bootstrap Full Sample$[b]$.

Calculate $\widehat{\text{BiasFactor}}_{2\text{SLS}}[b] = \left[\hat{s}_{\xi\zeta}\,\hat{s}_{\zeta\zeta}^{-1}\,\hat{s}'_{\xi\zeta}\right]^{-1}\hat{s}_{\xi\zeta}\,\hat{s}_{\zeta\zeta}^{-1}\,\hat{\psi}_{2SLS}$.

$d^{con}_{2\text{SLS}}[b] = (I_g - \widehat{\text{BiasFactor}}_{2\text{SLS}}[b])^{-1}d_{2\text{SLS}}[b]$.

end for

Infer test statistics from the distribution of $d^{con}_{2\text{SLS}}[b], \quad b = 1, \ldots, nB$.

Below is an example of how to implement this algorithm in Python:

```python
from joblib import Parallel, delayed  # For parallelization
import numpy as np
import pandas as pd
from linearmodels import IV2SLS

def bootstrap_2sls_corrected(calibration_df, full_df, ai_coded_regressors,
                             ai_coded_instruments, manual_regressors,
                             observed_regressors, dependent_variable,
                             n_boot=1000, n_jobs=-1, random_state=42):
    """
    Performs bootstrap resampling to estimate standard errors for the bias-corrected 2SLS estimator.

    Args:
        calibration_df (pd.DataFrame): DataFrame containing the calibration subsample.
        full_df (pd.DataFrame): DataFrame containing the full dataset.
        ai_coded_regressors (list): List of column names for AI-coded regressors.
        ai_coded_instruments (list): List of column names for AI-coded instruments.
        manual_regressors (list): List of column names for manually coded regressors.
```

```
        observed_regressors (list): List of column names for observed (non-AI-coded) regressors.
        dependent_variable (str): Name of the column containing the dependent variable.
        n_boot (int): Number of bootstrap iterations. Defaults to 1000.
        n_jobs (int): Number of CPU cores to use for parallelization. -1 means use all available cores.
                      Defaults to -1.
        random_state (int): Random seed for reproducibility. Defaults to 42.

    Returns:
        np.ndarray: A (n_boot x num_coefficients) array containing the bootstrap estimates of the coefficients.
    """

    np.random.seed(random_state)
    n_obs_full = full_df.shape[0]  # Number of observations in the full sample.
    n_obs_cal = calibration_df.shape[0]  # Number of observations in the calibration subsample.
    num_coefficients = len(ai_coded_regressors) + len(observed_regressors)  # Total number of coefficients.

    def single_bootstrap_iteration(i):
        """Performs a single bootstrap iteration."""
        # Resample with replacement from the calibration subsample
        cal_indices = np.random.choice(n_obs_cal, size=n_obs_cal, replace=True)
        boot_cal_df = calibration_df.iloc[cal_indices]

        # Resample with replacement from the full sample
        full_indices = np.random.choice(n_obs_full, size=n_obs_full, replace=True)
        boot_full_df = full_df.iloc[full_indices]

        # 1. Estimate moment matrices in the *resampled* calibration subsample
        s_xizeta_2sls, s_zetazeta_2sls, psi_hat_2sls = calculate_moment_matrices_2sls(
            boot_cal_df, ai_coded_regressors, ai_coded_instruments,
            manual_regressors, observed_regressors
        )

```

```
        # 2. Compute the canonical 2SLS estimator in the *resampled* full sample
        y = boot_full_df[dependent_variable].values
        xi_hat = boot_full_df[ai_coded_regressors + observed_regressors].values
        zeta_hat = boot_full_df[ai_coded_instruments].values

        model_2sls = IV2SLS(y, xi_hat, None, zeta_hat)
        d_2sls = model_2sls.fit(cov_type='unadjusted').params.values  # Get the parameters as a NumPy array

        # 3. Construct A_hat_2sls and B_hat_2sls *using the resampled moment matrices*
        A_hat_2sls = s_xizeta_2sls @ np.linalg.inv(s_zetazeta_2sls) @ s_xizeta_2sls.T
        B_hat_2sls = s_xizeta_2sls @ np.linalg.inv(s_zetazeta_2sls) @ psi_hat_2sls

        # 4. Solve for the corrected estimator d_con_2sls
        I_g = np.eye(xi_hat.shape[1])  # Identity matrix
        d_con_2sls = np.linalg.solve(I_g - np.linalg.inv(A_hat_2sls) @ B_hat_2sls, d_2sls)

        return d_con_2sls

    # Parallelize the bootstrap iterations
    bootstrap_results = Parallel(n_jobs=n_jobs)(
        delayed(single_bootstrap_iteration)(i) for i in range(n_boot)
    )

    return np.array(bootstrap_results)


# --- Example Usage ---
# Assuming you have 'calibration_df' and 'full_df' DataFrames,
# and you've defined the lists of column names:
# ai_coded_regressors, ai_coded_instruments, manual_regressors, observed_regressors, dependent_variable

# Run the bootstrap
# bootstrap_estimates = bootstrap_2sls_corrected(
#     calibration_df, full_df, ai_coded_regressors, ai_coded_instruments,
#     manual_regressors, observed_regressors, 'dependent_var_name',  # Replace 'dependent_var_name'
```

```
#     n_boot=1000, n_jobs=-1
# )

# Calculate standard errors
# bootstrap_se = np.std(bootstrap_estimates, axis=0)
# print("Bootstrap Standard Errors:", bootstrap_se)

# Calculate 95% confidence intervals (percentile method)
# bootstrap_ci_lower = np.percentile(bootstrap_estimates, 2.5, axis=0)
# bootstrap_ci_upper = np.percentile(bootstrap_estimates, 97.5, axis=0)
# print("95% Confidence Intervals (Lower):", bootstrap_ci_lower)
# print("95% Confidence Intervals (Upper):", bootstrap_ci_upper)
```

B.4.2 System-of-Equations Models (3SLS)

In the system-of-equations context, we consider models of the form:

$$Y = \Xi\Delta + E, \tag{13}$$

where $Y = \text{vec}(y_1, \ldots, y_G)$ is the vector of dependent variables, $\Xi = \text{diag}(\xi_1, \ldots, \xi_G)$ is the block-diagonal matrix of regressors, $\Delta = \text{vec}(\delta_1, \ldots, \delta_G)$ is the vector of coefficients, and $E = \text{vec}(\varepsilon_1, \ldots, \varepsilon_G)$ is the vector of error terms. We assume that the errors may be correlated across equations, with $\text{Var}(E) = \Sigma \otimes I_N$, where Σ is the $G \times G$ covariance matrix of the errors.

The proposed bias-corrected estimator for the system-of-equations case, denoted by $D^{\text{con}}_{\text{3SLS}}$, is given by the solution to:

$$\left[I_{Gg} - \hat{A}^{-1}_{\text{3SLS}}\hat{B}_{\text{3SLS}}\right] D^{\text{con}}_{\text{3SLS}} = D_{\text{3SLS}}, \tag{14}$$

where D_{3SLS} is the canonical 3SLS estimator, and $\hat{A}_{\text{3SLS}}$ and $\hat{B}_{\text{3SLS}}$ are estimates of the matrices defined in Web Appendix D (Equation (D.9) or (D.14) if the instruments vary across equations).

We use Algorithm 2 from Web Appendix D, now with updated notation:

Here's an example of how to implement this algorithm in Python, assuming a system of *two equations*:

Algorithm 5 Algorithm for Inference in System-of-Equations Marketing Models

for $b \leftarrow 1, nB$ **do**

Draw bootstrap sample Bootstrap Subsample$[b]$ of size N_s from Data$_s$.

Infer $\hat{\psi}_{g,3SLS} = \frac{\vec{\zeta}' \eta_{\xi g}}{N_s}, g = 1, \ldots G$ in Bootstrap Subsample$[b]$.

Draw bootstrap sample Bootstrap Full Sample$[b]$ of size N_f from Data$_f$.

Compute the 3SLS estimator $D_{3\text{SLS}}[b]$ in Bootstrap Full Sample$[b]$.

Infer $\hat{s}_{\xi_g \zeta, 3SLS}$ for $g = 1, \ldots, G$ and $\hat{s}_{\zeta\zeta, 3SLS}$ in Bootstrap Full Sample$[b]$.

Infer the estimation analogs of $A_{3\text{SLS}}$ and $B_{3\text{SLS}}$: $\hat{A}_{3\text{SLS}}$ and $\hat{B}_{3\text{SLS}}$.

Solve $\left[I_{Gg} - \hat{A}^{-1}_{3\text{SLS}} \hat{B}_{3\text{SLS}}\right] D^{\text{con}}_{3\text{SLS}}[b] = D_{3\text{SLS}}[b]$ in Bootstrap Full Sample$[b]$.

end for

Infer test statistics from the distribution of $D^{\text{con}}_{3\text{SLS}}[b], \quad b = 1, \ldots, nB$.

```python
from joblib import Parallel, delayed
import numpy as np
import pandas as pd
from linearmodels import IV3SLS, IV2SLS  # We'll use IV2SLS for the first stage

def bootstrap_3sls_corrected(calibration_df, full_df,
                             eq1_ai_regressors, eq1_observed_regressors, eq1_manual_regressors,
                             eq2_ai_regressors, eq2_observed_regressors, eq2_manual_regressors,
                             instruments, eq1_dependent, eq2_dependent,
                             n_boot=1000, n_jobs=-1, random_state = 42):
    """
    Performs bootstrap resampling to estimate standard errors for the bias-corrected 3SLS estimator (two equations).

    Args:
        calibration_df (pd.DataFrame): DataFrame containing the calibration subsample.
        full_df (pd.DataFrame): DataFrame containing the full dataset.
        eq1_ai_regressors (list): Column names for AI-coded regressors in equation 1.
        eq1_observed_regressors (list): Column names for observed regressors in equation 1.
        eq1_manual_regressors (list): Column names for manually coded regressors in equation 1.
```

```
        eq2_ai_regressors (list): Column names for AI-coded regressors in equation 2.
        eq2_observed_regressors (list): Column names for observed regressors in equation 2.
        eq2_manual_regressors (list): Column names for manually coded regressors in equation 2.
        instruments (list): Column names for instruments (common to both equations).
        eq1_dependent (str): Name of the dependent variable column for equation 1.
        eq2_dependent (str): Name of the dependent variable column for equation 2.
        n_boot (int): Number of bootstrap iterations. Defaults to 1000.
        n_jobs (int): Number of CPU cores to use for parallelization (-1 for all). Defaults to -1.
        random_state (int): Random seed for reproducibility. Defaults to 42.

    Returns:
        np.ndarray: A (n_boot x num_coefficients) array containing the bootstrap estimates of the coefficients.
    """

    np.random.seed(random_state)
    n_obs_full = full_df.shape[0]  # Number of observations in the full sample.
    n_obs_cal = calibration_df.shape[0]  # Number of observations in the calibration subsample.
    num_coefficients = (len(eq1_ai_regressors) + len(eq1_observed_regressors) +
                        len(eq2_ai_regressors) + len(eq2_observed_regressors))

    def single_bootstrap_iteration(i):
        """Performs a single bootstrap iteration."""
        # Resample with replacement
        cal_indices = np.random.choice(n_obs_cal, size=n_obs_cal, replace=True)
        boot_cal_df = calibration_df.iloc[cal_indices]
        full_indices = np.random.choice(n_obs_full, size=n_obs_full, replace=True)
        boot_full_df = full_df.iloc[full_indices]

        # 1. Estimate moment matrices in the *resampled* calibration subsample
```

```
        s_xi1zeta_3sls, s_zetazeta_3sls, psi1_hat_3sls =
        ↪  calculate_moment_matrices_3sls(
            boot_cal_df,
            [{'ai_coded_regressors': eq1_ai_regressors,
              'manual_regressors': eq1_manual_regressors,
              'observed_regressors': eq1_observed_regressors}],
            instruments
        )[0][0], calculate_moment_matrices_3sls(
            boot_cal_df,
            [{'ai_coded_regressors': eq1_ai_regressors,
              'manual_regressors': eq1_manual_regressors,
              'observed_regressors': eq1_observed_regressors}],
            instruments
        )[1], calculate_moment_matrices_3sls(
            boot_cal_df,
            [{'ai_coded_regressors': eq1_ai_regressors,
              'manual_regressors': eq1_manual_regressors,
              'observed_regressors': eq1_observed_regressors}],
            instruments
        )[2][0] # Extract single values correctly

        s_xi2zeta_3sls, _, psi2_hat_3sls =
        ↪  calculate_moment_matrices_3sls(
            boot_cal_df,
            [{'ai_coded_regressors': eq2_ai_regressors,
              'manual_regressors': eq2_manual_regressors,
              'observed_regressors': eq2_observed_regressors}],
            instruments
        )[0][0], calculate_moment_matrices_3sls(
            boot_cal_df,
            [{'ai_coded_regressors': eq2_ai_regressors,
              'manual_regressors': eq2_manual_regressors,
              'observed_regressors': eq2_observed_regressors}],
            instruments
        )[1], calculate_moment_matrices_3sls(
            boot_cal_df,
            [{'ai_coded_regressors': eq2_ai_regressors,
              'manual_regressors': eq2_manual_regressors,
              'observed_regressors': eq2_observed_regressors}],
            instruments
        )[2][0]  # Extract single values correctly


        # 2. Estimate Sigma_hat using 2SLS residuals (from the
        ↪  *bootstrapped* full sample)
```

```
        resids_2sls = []  # Renamed for clarity
        for y_col, ai_cols, obs_cols in zip([eq1_dependent,
        ↪  eq2_dependent],
                                            [eq1_ai_regressors,
                                            ↪  eq2_ai_regressors],
                                            [eq1_observed_regressors,
                                            ↪  eq2_observed_regressor
                                            ↪  s]):
            y = boot_full_df[y_col].values
            xi_hat = boot_full_df[ai_cols + obs_cols].values
            zeta_hat = boot_full_df[instruments].values
            model_2sls_i = IV2SLS(y, xi_hat, None, zeta_hat) # Renamed
            ↪  for clarity
            resids_2sls_i =
            ↪  model_2sls_i.fit(cov_type='unadjusted').resids.values #
            ↪  Extract residuals, renamed
            resids_2sls.append(resids_2sls_i)
        resids_2sls = np.column_stack(resids_2sls)
        Sigma_hat_3sls = (resids_2sls.T @ resids_2sls) /
        ↪  resids_2sls.shape[0] # Renamed

        # 3. Compute the canonical 3SLS estimator (using the
        ↪  *bootstrapped* full sample)
        y1 = boot_full_df[eq1_dependent].values
        y2 = boot_full_df[eq2_dependent].values
        xi1_hat = boot_full_df[eq1_ai_regressors +
        ↪  eq1_observed_regressors].values
        xi2_hat = boot_full_df[eq2_ai_regressors +
        ↪  eq2_observed_regressors].values
        zeta_hat = boot_full_df[instruments].values # Common
        ↪  instruments

        system = {'eq1': {'endog': y1, 'exog': xi1_hat, 'instruments':
        ↪  zeta_hat},
                  'eq2': {'endog': y2, 'exog': xi2_hat, 'instruments':
                  ↪  zeta_hat}}
        model_3sls = IV3SLS(system) # Renamed
        D_3sls = model_3sls.fit(cov_type='unadjusted').params.values #
        ↪  Extract parameters, renamed

        # 4. Construct A_hat_3sls and B_hat_3sls *using the resampled
        ↪  moment matrices*
        Sigma_inv_3sls = np.linalg.inv(Sigma_hat_3sls) # Renamed

```

```
        # Corrected A_hat_3sls: Block diagonal with per-equation terms
        A_hat_inv_eq1 = s_xi1zeta_3sls @ np.linalg.inv(s_zetazeta_3sls) @ s_xi1zeta_3sls.T
        A_hat_inv_eq2 = s_xi2zeta_3sls @ np.linalg.inv(s_zetazeta_3sls) @ s_xi2zeta_3sls.T
        A_hat_3sls = np.kron(Sigma_inv_3sls, np.block([[A_hat_inv_eq1, np.zeros_like(A_hat_inv_eq1)],
                                                       [np.zeros_like(A_hat_inv_eq2), A_hat_inv_eq2]])) # Renamed

        # Corrected B_hat_3sls: Block diagonal Psi_hat and per-equation s_xizeta
        B_hat_top = s_xi1zeta_3sls @ np.linalg.inv(s_zetazeta_3sls) @ psi1_hat_3sls
        B_hat_bottom = s_xi2zeta_3sls @ np.linalg.inv(s_zetazeta_3sls) @ psi2_hat_3sls
        B_hat_3sls = np.kron(Sigma_inv_3sls, np.block([[B_hat_top, np.zeros_like(B_hat_top)],
                                                       [np.zeros_like(B_hat_bottom), B_hat_bottom]])) # Renamed

        # 5. Solve for the corrected estimator D_con_3sls
        I_Gg = np.eye(xi1_hat.shape[1] + xi2_hat.shape[1])  # Identity matrix.
        D_con_3sls = np.linalg.solve(I_Gg - np.linalg.inv(A_hat_3sls) @ B_hat_3sls, D_3sls) # Renamed

        return D_con_3sls

    # Parallelize the bootstrap iterations
    bootstrap_results = Parallel(n_jobs=n_jobs)(
        delayed(single_bootstrap_iteration)(i) for i in range(n_boot)
    )

    return np.array(bootstrap_results)

# --- Example Usage ---
#  (Assuming appropriate DataFrame and column names)
# bootstrap_estimates = bootstrap_3sls_corrected(
#     calibration_df, full_df,
```

```
#     eq1_ai_regressors=['z_nost_ai'],
#     eq1_observed_regressors=['x_fea'],
#     eq1_manual_regressors=['z_nost_manual'],
#     eq2_ai_regressors=['z_price_ai'],
#     eq2_observed_regressors=['x_quality'],
#     eq2_manual_regressors=['z_price_manual'],
#     instruments=['w_her_ai', 'w_cost_ai'],
#     eq1_dependent='y1',
#     eq2_dependent='y2',
#     n_boot=1000, n_jobs=-1
# )

# # Calculate standard errors and confidence intervals (same as 2SLS
 ↪  example)
# bootstrap_se = np.std(bootstrap_estimates, axis=0)
# bootstrap_ci_lower = np.percentile(bootstrap_estimates, 2.5, axis=0)
# bootstrap_ci_upper = np.percentile(bootstrap_estimates, 97.5, axis=0)
```

B.5 Addressing Imprecision in Manually Coded Measures

The previous sections of this guide have assumed that the manually coded data in the calibration subsample represents a perfect "ground truth." While manual coding, particularly with multiple coders and careful protocols, aims for high accuracy, some degree of imprecision is almost inevitable, especially when dealing with subjective constructs like nostalgia, humor, or physical appeal. Individual coders may interpret the coding scales differently, leading to variability in the ratings. This imprecision in the manually coded data can, in turn, affect the accuracy of our bias-corrected estimators. Specifically, it can introduce bias into the estimation of the moment matrix $\hat{\psi}_{2SLS}$ (or $\hat{\psi}_{g,3SLS}$ for system-of-equations models), which captures the relationship between the AI-coding errors and the instruments. If $\hat{\psi}_{2SLS}$ (or $\hat{\psi}_{g,3SLS}$) is biased, the final corrected estimates ($d^{\text{con}}_{\text{2SLS}}$ or $D^{\text{con}}_{\text{3SLS}}$) will also be biased.

To address this issue, we adapt the SIMEX method. SIMEX is a technique designed to correct for measurement error in explanatory variables. The core idea is to:

1. **Simulation:** Systematically *add* known amounts of additional noise to the already-imprecise

manually coded variables.

2. **Estimation:** Re-estimate the model parameters (including the moment matrices and the final coefficients) for each level of added noise.
3. **Extrapolation:** Model the relationship between the added noise and the estimated parameters, and then *extrapolate* back to the case of *no* added noise. This extrapolated value is our corrected estimate.

This subsection provides a step-by-step guide to implementing SIMEX within our overall framework.

B.5.1 Step-by-Step SIMEX Procedure

1. **Define the Imprecision Parameter (λ):**

   We introduce a parameter, λ, to control the amount of synthetic noise added to the manually coded variables. $\lambda = 0$ represents no added noise (i.e., using the original manually coded data as is). $\lambda = 1$ represents adding noise with a standard deviation equal to the *estimated* standard deviation of the manual coding error itself. Values of $\lambda > 1$ represent adding even more noise. We will use a range of λ values to simulate increasing levels of imprecision.

2. **Estimate the Imprecision in the Manual Coding (λ_{i*}):**

   A crucial step is to estimate the inherent imprecision in the *original* manually coded data. This gives us a baseline level of noise. We have two options:

   - **Option 1 (Recommended):** If you have multiple coders rating each item (as strongly recommended in Section 1), you can use the *variation across coders* as a proxy for the imprecision. Specifically, we calculate the *average within-item variance* across all items in the calibration subsample. This average variance is an estimate of the variance of the manual coding error. We then take the square root to obtain the standard deviation. This is the preferred method because it directly uses the available data to estimate the noise.

- **Option 2 (Less Recommended):** If you *don't* have multiple coders per item, you'll need to make an *assumption* about the level of imprecision. For example, you might assume that the standard deviation of the manual coding error is a certain percentage (e.g., 10%, 20%) of the standard deviation of the manually coded variable itself. This is much less reliable than Option 1.

We *strongly* recommend using Option 1 whenever possible. The following code demonstrates how to calculate the average within-item variance and the corresponding standard deviation (Option 1):

```python
import numpy as np
import pandas as pd

def estimate_manual_coding_sd(calibration_df: pd.DataFrame, manual_cols: list) -> float:
    """
    Estimates the standard deviation of manual coding error when multiple coders
    rate each item.

    Args:
        calibration_df: DataFrame containing the calibration data.
                                        Must include multiple columns for manual ratings.
        manual_cols: List of column names representing the manual ratings
                             from different coders.

    Returns:
        The estimated standard deviation of the manual coding error.
    """

    # Calculate the within-item variance for each item.
    within_item_variances = calibration_df[manual_cols].var(axis=1)

    # Calculate the average within-item variance across all items.
```

```
    average_within_item_variance = within_item_variances.mean()

    # Take the square root to get the standard deviation.
    estimated_sd = np.sqrt(average_within_item_variance)

    return estimated_sd

# --- Example Usage ---
# Create a sample DataFrame (replace with your actual data)
data = {
    'Ad ID': range(1, 101),
    'coder1': np.random.randint(1, 8, size=100),  # Ratings from coder 1
    'coder2': np.random.randint(1, 8, size=100),  # Ratings from coder 2
    'coder3': np.random.randint(1, 8, size=100),  # Ratings from coder 3
    'z_nost_ai': np.random.rand(100) * 7,
    'x_fea': np.random.rand(100),
    'w_her_ai': np.random.rand(100),
    'w_cost_ai': np.random.rand(100)
}
calibration_df = pd.DataFrame(data)
manual_cols = ['coder1', 'coder2', 'coder3']

# Estimate the standard deviation of manual coding error
manual_coding_sd = estimate_manual_coding_sd(calibration_df, manual_cols)
print(f"Estimated SD of manual coding error: {manual_coding_sd:.3f}")
```

We also need to calculate the standard deviation of the manually coded variable itself. We will use this to calculate the estimated value of λ_{i*}. This is calculated as the estimated standard deviation of the manual coding error (from `estimate_manual_coding_sd`) divided by the standard deviation of the manually coded variable (which can be calculated using `np.std` on any of the `manual_cols`, or an average of them).

3. **Create a Grid of λ Values:**

   Next, we create a sequence of λ values. This grid should start at or above our estimated

impreciseion level (λ_{i*} or its equivalent) and extend to higher values. For instance, if we estimate the standard deviation of the manual coding error to be 0.5, and the standard deviation of the manually coded variable to be 1.0, then our estimated λ_{i*} is 0.5. We might create a grid like [0.5, 0.6, 0.7, ..., 1.5, 2.0]. The reasoning is that we want to simulate the effect of *increasing* levels of noise on our estimates.

The following code demonstrates how to create a grid of λ values:

```
import numpy as np

def create_lambda_grid(estimated_lambda_star: float, num_points: int = 20, max_lambda_factor: float = 2.0) -> np.ndarray:
    """
    Creates a grid of lambda values for the SIMEX procedure.

    Args:
        estimated_lambda_star:  The estimated value of lambda_i*. This is
                                the estimated SD of manual coding error / SD of manual variable.
        num_points: The number of points in the grid.  Defaults to 20.
        max_lambda_factor:  The maximum lambda value, expressed as a multiple of
                            estimated_lambda_star.  Defaults to 2.0.

    Returns:
        A NumPy array of lambda values.
    """

    max_lambda = estimated_lambda_star * max_lambda_factor
    lambda_grid = np.linspace(estimated_lambda_star, max_lambda, num_points)
    return lambda_grid

# --- Example Usage ---
# Assuming you've already calculated manual_coding_sd and the SD of the manual variable
```

```
manual_variable_sd = calibration_df[manual_cols].values.std() # Use all manual columns
estimated_lambda_star = manual_coding_sd / manual_variable_sd
lambda_grid = create_lambda_grid(estimated_lambda_star)
print(f"Lambda grid: {lambda_grid}")
```

4. **Simulation Step (Loop):**

This is the core of the SIMEX procedure. For each λ value in our grid, we do the following:

- **Add Noise:** We add random, normally distributed noise to the *manually coded* variables in the calibration subsample. The noise has a mean of 0 and a standard deviation of λ times the estimated standard deviation of the manual coding error (calculated in Step 2). *It is crucial that this noise is added only to the manually coded variables, not the AI-coded variables.*
- **Re-estimate Moment Matrices:** We re-calculate the moment matrices ($\hat{s}_{\xi\zeta,2SLS}$, $\hat{s}_{\zeta\zeta,2SLS}$, and $\hat{\psi}_{2SLS}$ for 2SLS, or $\hat{s}_{\xi_g\zeta,3SLS}$, $\hat{s}_{\zeta\zeta,3SLS}$, and $\hat{\psi}_{g,3SLS}$ for 3SLS) using the *noise-added* calibration subsample. This uses the `calculate_moment_matrices_2sls` or `calculate_moment_matrices_3sls` function from Section 3.
- **Re-estimate Coefficients:** We re-calculate the bias-corrected estimator ($d^{\text{con}}_{\text{2SLS}}$ for single-equation models or $D^{\text{con}}_{\text{3SLS}}$ for system-of-equations models) using the *noise-added* moment matrices. This uses the same code as in Section 4, but with the updated moment matrices.
- **Store Results:** We store the estimated coefficients for this particular λ value.

Here's the Python code for a single simulation step (2SLS version).

```
def simex_simulation_step(calibration_df: pd.DataFrame, ai_coded_regressors: list, ai_coded_instruments: list,
                          manual_regressors: list, observed_regressors: list, dependent_variable: str,
                          lambda_value: float, manual_coding_sd: float, full_df: pd.DataFrame) -> np.ndarray:
```

```
    """
    Performs a single simulation step in the SIMEX procedure.

    Args:
        calibration_df: The calibration data.
        ai_coded_regressors: List of AI-coded regressor column names.
        ai_coded_instruments: List of AI-coded instrument column names.
        manual_regressors: List of manually coded regressor column names.
        observed_regressors: List of observed (non-AI-coded) regressor column names.
        dependent_variable: Name of dependent variable.
        lambda_value: The current lambda value (noise level).
        manual_coding_sd: Estimated SD of manual coding error.
        full_df: The full data.

    Returns:
        The bias-corrected 2SLS estimates for this simulation step (d_con_2sls).
    """

    # Create a copy to avoid modifying the original DataFrame
    noisy_cal_df = calibration_df.copy()

    # Add noise to the *manually coded* variables
    for manual_col in manual_regressors:
        noise = np.random.normal(loc=0, scale=lambda_value * manual_coding_sd, size=len(noisy_cal_df))
        noisy_cal_df[manual_col] = noisy_cal_df[manual_col] + noise

    # Re-estimate the moment matrices using the *noisy* calibration data.  Note the _2sls suffixes.
    s_xizeta_2sls, s_zetazeta_2sls, psi_hat_2sls = calculate_moment_matrices_2sls(
        noisy_cal_df, ai_coded_regressors, ai_coded_instruments,
        manual_regressors, observed_regressors
    )

    # Re-estimate the 2SLS model (using the *original*, full data - only the calibration sample is modified)
    y = full_df[dependent_variable].values
```

```
    xi_hat = full_df[ai_coded_regressors +
    ↪ observed_regressors].values
    zeta_hat = full_df[ai_coded_instruments].values
    model_2sls = IV2SLS(y, xi_hat, None, zeta_hat)
    d_2sls = model_2sls.fit(cov_type='unadjusted').params.values

    # Calculate the bias-corrected estimator.  RECALCULATE
    ↪ A_hat_2sls and B_hat_2sls here.
    A_hat_2sls = s_xizeta_2sls @ np.linalg.inv(s_zetazeta_2sls)
    ↪ @ s_xizeta_2sls.T
    B_hat_2sls = s_xizeta_2sls @ np.linalg.inv(s_zetazeta_2sls)
    ↪ @ psi_hat_2sls
    I_g = np.eye(xi_hat.shape[1])
    d_con_2sls = np.linalg.solve(I_g - np.linalg.inv(A_hat_2sls)
    ↪ @ B_hat_2sls, d_2sls)

    return d_con_2sls
```

5. **Extrapolation Step:**

   After completing the simulation step for all λ values, we have a set of estimated coefficients, one for each level of added noise. We now model the relationship between these estimated coefficients and the corresponding λ values. We can use a simple regression model (linear or quadratic) or a smoothing spline.

   The key idea is to *extrapolate* this relationship back to $\lambda = 0$, which represents the hypothetical case of *no added noise*. This extrapolated value is our final, SIMEX-corrected estimate.

   The following code demonstrates how to perform the extrapolation using `scipy.interpolate.inte`

```
from scipy.interpolate import interp1d

def simex_extrapolation(lambda_grid: np.ndarray, estimates:
↪ np.ndarray, extrapolation_type: str = 'linear') ->
↪ np.ndarray:
    """
    Performs the extrapolation step in the SIMEX procedure.
```

```
    Args:
        lambda_grid: The grid of lambda values used in the simulation.
        estimates: A (num_lambda_values x num_coefficients) array
                                containing the estimated coefficients for each lambda value.
        extrapolation_type:  The type of extrapolation to use ('linear', 'quadratic', or 'cubic').

    Returns:
        The extrapolated coefficient estimates at lambda = 0.
    """

    num_coefficients = estimates.shape[1]
    extrapolated_estimates = np.zeros(num_coefficients)

    for i in range(num_coefficients):
        # Fit an interpolation function for each coefficient
        if extrapolation_type == 'linear':
            f = interp1d(lambda_grid, estimates[:, i], kind='linear', fill_value="extrapolate")
        elif extrapolation_type == 'quadratic':
            f = interp1d(lambda_grid, estimates[:, i], kind='quadratic', fill_value="extrapolate")
        elif extrapolation_type == 'cubic':
            f = interp1d(lambda_grid, estimates[:, i], kind='cubic', fill_value="extrapolate")
        else:
            raise ValueError("Invalid extrapolation_type. Choose 'linear', 'quadratic', or 'cubic'.")

        # Extrapolate to lambda = 0
        extrapolated_estimates[i] = f(0)

    return extrapolated_estimates
```

6. **Standard Errors:**

To obtain standard errors for our SIMEX-corrected estimates, we embed the entire SIMEX

procedure *within* the bootstrap loop described in Section 4. That is, for each bootstrap iteration, we perform the full SIMEX procedure (Steps 1-5 above). This gives us a distribution of SIMEX-corrected estimates, from which we can calculate standard errors and confidence intervals, as before.

Here's the code that combines the bootstrap with the SIMEX procedure:

```
from joblib import Parallel, delayed
import numpy as np
import pandas as pd
from linearmodels import IV3SLS, IV2SLS  # Assuming these are available.
def perform_simex(calibration_df: pd.DataFrame, full_df: pd.DataFrame, ai_coded_regressors: list, ai_coded_instruments: list,
                  manual_regressors: list, observed_regressors: list, dependent_variable: str,
                  manual_cols: list, num_points: int = 20, max_lambda_factor: float = 2.0, extrapolation_type: str = 'linear',
                  n_boot: int = 1000, n_jobs: int = -1, random_state: int = 42) -> tuple:
    """
    Performs the complete SIMEX procedure, including bootstrapping, to correct for
    imprecision in manually coded variables.

    Args:
        calibration_df: DataFrame containing the calibration subsample.
        full_df: DataFrame containing the full dataset.
        ai_coded_regressors: List of column names for AI-coded regressors.
        ai_coded_instruments: List of column names for AI-coded instruments.
        manual_regressors: List of column names for manually coded regressors.
        observed_regressors: List of column names for observed (non-AI-coded) regressors.
        dependent_variable: Name of the dependent variable column.
        manual_cols: List of column names for the *multiple* manual ratings.
```

```
        num_points: Number of points in the lambda grid. Defaults to 20.
        max_lambda_factor: Maximum lambda value (multiple of estimated_lambda_star). Defaults to 2.0.
        extrapolation_type: 'linear', 'quadratic', or 'cubic'. Defaults to 'linear'.
        n_boot: Number of bootstrap iterations. Defaults to 1000.
        n_jobs: Number of CPU cores for parallelization (-1 for all). Defaults to -1.
        random_state: Random seed for reproducibility. Defaults to 42.

    Returns:
        tuple: (extrapolated_estimates, bootstrap_estimates, lambda_grid)
            extrapolated_estimates: The final SIMEX-corrected estimates.
            bootstrap_estimates: The raw bootstrap estimates (before extrapolation).
            lambda_grid: The grid of lambda values used.
    """
    np.random.seed(random_state)
    # 1. Estimate manual coding SD
    manual_coding_sd = estimate_manual_coding_sd(calibration_df, manual_cols)
    manual_variable_sd = calibration_df[manual_cols].values.std() # Use all manual columns
    estimated_lambda_star = manual_coding_sd / manual_variable_sd

    # 2. Create lambda grid
    lambda_grid = create_lambda_grid(estimated_lambda_star, num_points, max_lambda_factor)

    # 3. Bootstrap loop
    def single_bootstrap_iteration(i):
        # Standard bootstrap resampling
        np.random.seed(random_state*1000000 + i) # Important for reproducibility with parallelization!
        n_obs_full = full_df.shape[0]
        n_obs_cal = calibration_df.shape[0]
        cal_indices = np.random.choice(n_obs_cal, size=n_obs_cal, replace=True)
```

```
            boot_cal_df = calibration_df.iloc[cal_indices]
            full_indices = np.random.choice(n_obs_full,
             ↪  size=n_obs_full, replace=True)
            boot_full_df = full_df.iloc[full_indices]

            # SIMEX simulation steps *within* the bootstrap
             ↪  iteration
            simex_estimates = []
            for lambda_value in lambda_grid:
                d_con_2sls = simex_simulation_step( # Use d_con_2sls
                    boot_cal_df, ai_coded_regressors,
                     ↪  ai_coded_instruments,
                    manual_regressors, observed_regressors,
                     ↪  dependent_variable,
                    lambda_value, manual_coding_sd, boot_full_df
                )
                simex_estimates.append(d_con_2sls) # Append
                 ↪  d_con_2sls
            return np.array(simex_estimates)

        # Parallelize the bootstrap
        bootstrap_results = Parallel(n_jobs=n_jobs)(
            delayed(single_bootstrap_iteration)(i) for i in
             ↪  range(n_boot)
        )
        bootstrap_results = np.array(bootstrap_results)  # (n_boot,
         ↪  num_lambda, num_coefficients)

        # 4. Extrapolation (average over bootstrap iterations first)
        average_estimates = np.mean(bootstrap_results, axis=0)  #
         ↪  (num_lambda, num_coefficients)
        extrapolated_estimates = simex_extrapolation(lambda_grid,
         ↪  average_estimates, extrapolation_type)

        return extrapolated_estimates, bootstrap_results,
         ↪  lambda_grid # Return lambda_grid
```

B.5.2 Practical Considerations

- **Choice of Extrapolation Function:** The choice of extrapolation function (linear, quadratic, spline) can influence the results. It's generally recommended to start with a simple linear or

quadratic model. If the relationship between the estimates and λ appears non-linear, you might consider a cubic spline. However, avoid overly complex extrapolation functions, as they can be sensitive to noise. Experimentation and visual inspection of the extrapolation plots are recommended.

- **Range of λ Values:** The range of λ values should start at or above your estimated level of manual coding imprecision (λ_{i*}) and extend high enough to observe a clear trend. If the trend is not clear, you may need to increase the maximum λ value. It's generally not necessary to go beyond a `max_lambda_factor` of 2.0 (i.e., adding noise with twice the estimated standard deviation of the original manual coding error).
- **Computational Cost:** SIMEX, especially when combined with bootstrapping, can be computationally intensive. Parallelization (as shown in the code examples) is highly recommended.
- **Limitations:** SIMEX relies on the assumption that the relationship between the added noise and the estimated coefficients can be reasonably extrapolated back to zero. If this assumption is violated (e.g., if the relationship is highly non-linear even at low levels of noise), the SIMEX estimates may be biased. It is also important to remember that SIMEX addresses imprecision in the *manually coded* variables, not errors in the AI coding.

B.5.3 Hypothesis Testing and Inference

Once the bias-corrected estimators have been obtained (with or without SIMEX), researchers can conduct hypothesis tests and construct confidence intervals to draw inferences about the model parameters. The general approach is the same as with standard estimators, but with the corrected estimates and their associated standard errors used in place of the uncorrected values.

For example, to test the hypothesis that a particular coefficient δ_j is equal to zero, researchers can compute the t-statistic:

$$t = \frac{d^{\text{con}}_{\text{2SLS},j}}{\text{SE}(d^{\text{con}}_{\text{2SLS},j})}, \tag{15}$$

where $d^{\text{con}}_{\text{2SLS},j}$ is the bias-corrected estimate of δ_j and $\text{SE}(d^{\text{con}}_{\text{2SLS},j})$ is its estimated standard error (obtained from the bootstrap). Similarly, for the system of equations model,

$$t = \frac{D^{\text{con}}_{\text{3SLS},j}}{\text{SE}(D^{\text{con}}_{\text{3SLS},j})}, \tag{16}$$

Under the null hypothesis, this t-statistic follows a standard normal distribution (or a t-distribution with appropriate degrees of freedom), allowing for the computation of p-values and the construction of confidence intervals. The bootstrap provides the distribution of the estimates, from which standard errors and confidence intervals can be calculated using `np.std` for standard errors and `np.percentile` for confidence intervals.

In the system-of-equations context, researchers may also be interested in testing cross-equation restrictions or conducting joint tests of multiple coefficients. These tests can be performed using Wald tests or likelihood ratio tests, with the bias-corrected estimates and their covariance matrix used in place of the uncorrected values.

B.5.4 Model Diagnostics and Validation

After estimating the model, it is important to assess its goodness of fit and to check the validity of the underlying assumptions. Standard diagnostic tools, such as residual plots, tests for heteroskedasticity, and tests for overidentifying restrictions, can be used for this purpose. These diagnostics should be performed using the *corrected* estimates and residuals.

For example, in the single-equation context, researchers can plot the residuals against the fitted values or the regressors to check for patterns that might indicate model misspecification or non-constant error variance. In the system-of-equations context, they can examine the residuals from each equation and test for cross-equation correlation.

It is also important to validate the model's performance on out-of-sample data. This can be done by splitting the full sample into training and test sets, estimating the model (including the bias correction and, if applicable, SIMEX) on the training set, and then evaluating its predictive accuracy on the test set. If the calibration subsample is large enough, it can also be split to provide

an independent validation set for the AI models.

B.5.5 Computing Standard Errors via Bootstrap

To obtain valid standard errors for the bias-corrected estimators, we recommend using bootstrap methods. The bootstrap involves repeatedly resampling the data with replacement, re-estimating the model on each bootstrap sample, and then using the distribution of the resulting estimates to approximate the sampling distribution of the estimator. Code implementing the bootstrap for both the single equation and system of equations models is provided above.

The specific steps for implementing the bootstrap are outlined above. In general, the process involves the following:

1. Draw a random sample of size N_s with replacement from the calibration subsample.
2. Estimate the moment matrices $\hat{s}_{\xi\zeta,2SLS}$, $\hat{s}_{\zeta\zeta,2SLS}$, and $\hat{\psi}_{2SLS}$ (or the 3SLS equivalents) using this bootstrap sample.
3. Draw a random sample of size N with replacement from the full sample.
4. Compute the canonical estimator (OLS, 2SLS, or 3SLS) using this bootstrap sample.
5. Solve for the bias-corrected estimator using the bootstrapped moment matrices and the canonical estimator.
6. Repeat steps 1-5 a large number of times (e.g., 1,000) to obtain a distribution of bias-corrected estimates.
7. Compute the standard errors and confidence intervals from the quantiles of the bootstrap distribution.

The bootstrap can be computationally intensive, especially for large datasets or complex models. However, it provides a flexible and robust method for obtaining standard errors that does not rely on specific distributional assumptions about the errors. When SIMEX is used, the entire SIMEX procedure (including adding noise, re-estimating moment matrices, and extrapolating) is performed *within* each bootstrap iteration, as shown in the `perform_simex` function.

C Addressing AI-Coding Errors in Single-Equation Marketing Models

This section expands on the econometric framework introduced in the manuscript, focusing on single-equation models (i.e., models used in research studies where there is only one dependent variable). We begin with a motivating example, generalize to matrix notation, and explicitly characterize the structure of AI-coding errors. We then derive the bias that arises when these errors are ignored and develop a consistent estimator to address this bias. The notation and model setup directly follow from the manuscript.

Consider a study examining how humor in social media posts affects user engagement. The true relationship is:

$$\text{likes}_i = \alpha + \beta \cdot \text{humor}_i + \varepsilon_i, \tag{17}$$

where humor_i is a latent construct (e.g., humor intensity) extracted from unstructured data (e.g., video content). In practice, researchers only observe $\hat{\text{humor}}_i$, an AI-coded proxy for humor_i, which introduces measurement error:

$$\hat{\text{humor}}_i = \text{humor}_i + \eta_i. \tag{18}$$

As we show formally later in this section, naïvely substituting $\hat{\text{humor}}_i$ for humor_i in the regression leads to biased estimates of β. This bias arises because η_i may be correlated with humor_i (or other regressors, such as video quality in a more complex multivariate regression model), violating classical measurement error assumptions. Such a bias is also not mitigated by a high correlation between the AI-coded variable and the true variable—in our manuscript, we document biases even when the correlation between the true variable and the AI-coded counterpart is as high as 0.97 (for physical appeal in the Instagram study) and 0.90 (for humor in the YouTube study). This example illustrates the fundamental econometric issue this research addresses.

Recall from the manuscript the general form of the single-equation model:

$$y_i = \alpha + \sum_{m=1}^{M} \beta_m z_{mi} + \sum_{l=1}^{L} \gamma_l x_{li} + \varepsilon_i. \tag{19}$$

As in the manuscript, y_i is the dependent variable, z_{mi} are the regressors coded from unstructured data, x_{li} are the observed (non-coded) regressors, and ε_i is the error term. The index i denotes an individual observation, ranging from 1 to N, the total number of observations.

To facilitate a more compact and general derivation, we represent this model in matrix form:

$$y = \xi\delta + \varepsilon. \tag{20}$$

Where:

- y is an $N \times 1$ vector of dependent variable observations.
- $\xi = [z\ x]$ is an $N \times g$ *true* design matrix of the regressors, where z is the $N \times M$ matrix of *true* (uncoded) values of the AI-coded regressors, and x is the $N \times L$ matrix of observed regressors. Thus, this matrix includes *both* the true values of the AI-coded regressors (corresponding to the z_{mi} variables in Equation (19)) *and* the observed regressors (corresponding to the x_{li} variables in Equation (19)).
- δ is a $g \times 1$ vector of coefficients, where $g = 1 + M + L$ (including the intercept). This vector concatenates the intercept (α), the coefficients on the AI-coded regressors (β_m), and the coefficients on the observed regressors (γ_l). Thus, $\delta = [\alpha, \beta', \gamma']'$.
- ε is an $N \times 1$ vector of error terms.

Analogously, ζ is the $N \times h$ *true* instrument matrix with $h \geq g$. ζ includes *all* instruments: (1) any exogenous regressors from x that are also valid instruments, (2) the *true* values of any exogenous regressors from z that are also valid instruments, and (3) the *true* values of any variables, w, that are used *only* as instruments (and are not included in x or z). We assume that $E[\varepsilon_i|\zeta] = 0$ (exogeneity with respect to the *true* instruments) and that the conditional variance is

$\text{Var}(\varepsilon_i \mid \zeta) = \sigma^2_{\varepsilon,i}$ (accommodating heteroskedasticity, meaning the variance of the error term can depend on the regressors).

z is *unobserved* in the full sample; we only observe its AI-coded counterpart, $\hat{z}$. Therefore, we define $\hat{\xi} = [\hat{z}\ x]$, the observed design matrix, and $\hat{\zeta}$ is the observed instrument matrix. These matrices are analogous to the true matrices ξ and ζ, respectively, except that the *true* values of the AI-coded variables are replaced by their *AI-coded* counterparts. The AI-coding process introduces errors for both regressors and instruments:

$$\hat{z}_m = z_m + \eta_{zm}, \quad \hat{w}_p = w_p + \eta_{wp}, \tag{21}$$

where $\hat{z}_m$ is the AI-coded value and η_{zm} is the AI-coding error for the m-th regressor, and $\hat{w}_p$ is the AI-coded value and η_{wp} is the AI-coding error for the p-th instrument. In matrix notation, $\hat{z} = z + \eta_z$ and $\hat{w} = w + \eta_w$, where η_z is the $N \times M$ matrix with elements η_{zm}. The matrix of coding errors for all regressors is $\eta_\xi = [\eta_z\ 0]$, where 0 is an $N \times L$ matrix of zeros (since the observed regressors, x, have no coding error). Similarly, η_ζ is the matrix of coding errors for all instruments, where all columns that are observed and not coded with AI are 0.

This nonclassical error structure invalidates conventional estimators like OLS or 2SLS that assume $\eta_\xi, \eta_\zeta \perp\!\!\!\perp \xi, \zeta$. However, the calibration subsample allows us to model $\mathbb{E}[\eta_z|\xi]$ and $\mathbb{E}[\eta_w|\xi]$ directly, circumventing the need for latent variable extrapolation.

We derive the bias arising from the naïve application of the 2SLS estimator to AI-coded variables:

$$d_{\text{2SLS}} = \left[\hat{\xi}'\hat{\zeta}\left(\hat{\zeta}'\hat{\zeta}\right)^{-1}\hat{\zeta}'\hat{\xi}\right]^{-1} \underbrace{\hat{\xi}\hat{\zeta}\left(\hat{\zeta}'\hat{\zeta}\right)^{-1}\hat{\zeta}'}_{:=\overline{P}_{\text{2SLS}}} y = \left[\hat{\xi}'\overline{P}_{\text{2SLS}}\hat{\xi}\right]^{-1}\hat{\xi}'\overline{P}_{\text{2SLS}}y, \tag{22}$$

where $\overline{p}_{2\text{SLS}} = \hat{\zeta}(\hat{\zeta}'\hat{\zeta})^{-1}\hat{\zeta}'$ is the projection matrix. The asymptotic bias is:

$$
\begin{aligned}
\operatorname*{plim}_{N\to\infty} d_{2\text{SLS}} &= \operatorname*{plim}_{N\to\infty}\left[\hat{\xi}'\hat{\zeta}\left(\hat{\zeta}'\hat{\zeta}\right)^{-1}\hat{\zeta}'\hat{\xi}\right]^{-1}\hat{\xi}'\hat{\zeta}\left(\hat{\zeta}'\hat{\zeta}\right)^{-1}\hat{\zeta}'y \\
&= \delta + \operatorname*{plim}_{N\to\infty}\left[\hat{\xi}'\hat{\zeta}\left(\hat{\zeta}'\hat{\zeta}\right)^{-1}\hat{\zeta}'\hat{\xi}\right]^{-1}\hat{\xi}'\hat{\zeta}\left(\hat{\zeta}'\hat{\zeta}\right)^{-1}\hat{\zeta}'u \\
&= \delta + \left[s_{\xi\zeta}s_{\zeta\zeta}^{-1}s_{\xi\zeta}'\right]^{-1}s_{\xi\zeta}s_{\zeta\zeta}^{-1}\operatorname*{plim}_{N\to\infty}\frac{\hat{\zeta}'u}{N} \\
&= \delta + \left[s_{\xi\zeta}s_{\zeta\zeta}^{-1}s_{\xi\zeta}'\right]^{-1}s_{\xi\zeta}s_{\zeta\zeta}^{-1}\operatorname*{plim}_{N\to\infty}\frac{\hat{\zeta}'(\varepsilon-\eta_\xi\delta)}{N} \\
&= \delta + \left[s_{\xi\zeta}s_{\zeta\zeta}^{-1}s_{\xi\zeta}'\right]^{-1}s_{\xi\zeta}s_{\zeta\zeta}^{-1}\operatorname*{plim}_{N\to\infty}\frac{\zeta'\varepsilon+\eta_\zeta'\varepsilon-\hat{\zeta}'\eta_\xi\delta}{N} \\
&= \delta - \underbrace{\left[s_{\xi\zeta}s_{\zeta\zeta}^{-1}s_{\xi\zeta}'\right]^{-1}s_{\xi\zeta}s_{\zeta\zeta}^{-1}\psi_{2\text{SLS}}\delta}_{\text{bias}},
\end{aligned}
\tag{23}
$$

where $y = \left(\hat{\xi} - \eta_\xi\right)\delta + \varepsilon = \hat{\xi}\delta + u$, and $u = \varepsilon - \eta_\xi\delta$ is the composite error, $s_{\xi\zeta} = \operatorname{plim}_{N\to\infty}\frac{\hat{\xi}'\hat{\zeta}}{N}$, $s_{\zeta\zeta} = \operatorname{plim}_{N\to\infty}\frac{\hat{\zeta}'\hat{\zeta}}{N}$, $\psi_{2\text{SLS}} = \operatorname{plim}_{N\to\infty}\frac{\hat{\zeta}'\eta_\xi}{N}$. By construction, the structural errors are orthogonal to the true values of the instruments: $\operatorname{plim}_{N\to\infty}\frac{\zeta'\varepsilon}{N} = 0$. Moreover, we assume that the structural errors are orthogonal to the coding errors $\operatorname{plim}_{N\to\infty}\frac{\eta_\zeta'\varepsilon}{N} = 0$. $s_{\xi\zeta}$ and $s_{\zeta\zeta}$ are the probability limits of the sample cross-product matrices of the *observed* regressors and instruments. $\psi_{2\text{SLS}}$ is an $h \times g$ matrix representing the probability limit of the sample cross-product matrix of the *observed* instruments and the coding errors.

The 2SLS estimator is consistent if and only if $\psi_{2\text{SLS}} = 0$. While this condition is theoretically possible, it requires a precise cancellation between the cross-covariance of the true values and the coding errors and the self-covariance of the coding errors. In any realistic research setting—with typical data variation and AI-coding methods—such an exact cancellation is virtually implausible, so that $\psi_{2\text{SLS}}$ is almost always nonzero, leading to bias when coding errors are ignored.

These results are novel to the literature as our model structure is more complex than prior research. Specifically, marketing models often include endogenous variables such as price, which are associated with a high likelihood of being simultaneously determined or otherwise correlated

with the structural error due to omitted variables. Consequently, it is common in marketing to employ instruments—variables that are correlated with the endogenous variables but not with the structural error—in field studies. When setting our framework and estimator, we explicitly consider cases where (1) an endogenous variable is observed and therefore a part of x, (2) an endogenous variable is AI-coded and therefore a part of $\hat{z}$, and (3) an instrument is AI-coded and therefore a part of $\hat{w}$.

In contrast, existing literature on measurement error assumes the *true* values of all regressors are exogenous (i.e., $E[\varepsilon \mid \xi] = 0$) and that all instruments are observed (i.e., $\eta_w = 0$). Moreover, the term 'instrument' in this literature is reserved for the treatment of measurement error and not the treatment of other classical sources of endogeneity such as simultaneity and omitted variables that are common concerns in marketing research; existing models typically assume that the instruments are orthogonal to the coding errors, i.e., $E[\eta_\xi \mid \zeta] = 0$, and that the instruments are observed without error (i.e., $\eta_w = 0$). These assumptions are relaxed in our model structure to align it with AI-coded regressors and instruments where both AI-coding error is likely to be correlated with other data features and AI is likely to be used to develop instruments, or instruments developed by averaging AI-coded variables (such as using the mean of AI-coded prices in an adjacent market as an instrument for AI-coded price in a market). These features particularly distinguish our research and model structure.

C.1 Bias Direction

Under classical measurement error assumptions (uncorrelated errors and exogenous variables), the magnitude of the true relationship is typically underestimated—a phenomenon known as attenuation bias, downward bias, or dilution bias (Aydemir and Borjas 2011).[10] This implies that theory development using measured-with-error variables is only susceptible to Type II errors (i.e., failing to reject a false null hypothesis), which are often viewed as less damaging than Type I

[10] Jerry Hausman famously noted: "At MIT I have called this the 'Iron Law of Econometrics'—the magnitude of the estimate is usually smaller than expected. It is also called 'attenuation' in the statistics literature." (Hausman 2001, p. 58).

errors (i.e., rejecting a true null hypothesis, leading to statistical support for incorrect theory).

However, with AI-coded variables, this simple intuition may not hold. The direction and magnitude of the bias depend on the complex interplay between the coding errors, the regressors, and the instruments. To illustrate this, consider a simplified scenario where only the first regressor is AI-coded (and therefore subject to coding error), while all other regressors and instruments are observed without error. In this case, the matrix ψ_{2SLS} (representing the probability limit of the cross-product of observed instruments and coding errors) will have the following structure:

- $\psi_{\text{2SLS}}[j] = 0$ for $j = 2, \ldots, g$ (where $\psi_{\text{2SLS}}[j]$ denotes the j-th column of ψ_{2SLS}). This is because the coding errors for all other observed regressors are zero.
- $\psi_{\text{2SLS}}[1] \neq 0$, reflecting the potential correlation between the AI-coding errors of the first regressor and the instruments.

Let $\delta[i]$ denote the i-th element of the true coefficient vector δ. From Equation (23), the asymptotic bias of the i-th coefficient in the uncorrected 2SLS estimator is given by:

$$\text{Bias}(\operatorname*{plim}_{N\to\infty} d_{\text{2SLS}}[i]) = -\left[\left(s_{\xi\zeta} s_{\zeta\zeta}^{-1} s_{\xi\zeta}'\right)^{-1} s_{\xi\zeta} s_{\zeta\zeta}^{-1} \psi_{\text{2SLS}} \delta\right][i],, \tag{24}$$

where $[A][i]$ denotes the i-th element of vector A. When only the first regressor is AI-coded, this simplifies to:

$$\text{Bias}(\operatorname*{plim}_{N\to\infty} d_{\text{2SLS}}[i]) = -\left[\left(s_{\xi\zeta} s_{\zeta\zeta}^{-1} s_{\xi\zeta}'\right)^{-1} s_{\xi\zeta} s_{\zeta\zeta}^{-1} \psi_{\text{2SLS}}[1]\right][i]\delta[1]. \tag{25}$$

The sign of this bias depends on the sign of $\delta[1]$ and the sign of the i-th element of the vector $\left(s_{\xi\zeta} s_{\zeta\zeta}^{-1} s_{\xi\zeta}'\right)^{-1} s_{\xi\zeta} s_{\zeta\zeta}^{-1} \psi_{\text{2SLS}}[1]$. This latter term is a complex function of the relationships between the true regressors, the instruments, and the AI-coding errors. It is *not* guaranteed to be positive.

As such, the bias can be *either* positive *or* negative, even for the coefficient of the AI-coded variable itself ($\delta[1]$). Furthermore, the bias can *propagate* to the coefficients of the *observed* variables (i.e., $d_{\text{2SLS}}[i]$ for $i > 1$) due to the correlations between the AI-coded regressor and the

other regressors. In other words, even if the AI-coded variable enters the model *solely* as a control, its inclusion can bias the estimates of other key variables—even those that are observed without any coding error.

This demonstrates that AI-coding error can lead to *both* attenuation *and* amplification of coefficient estimates, and the direction of the bias is not generally predictable without further information. While the matrix product $\left(s_{\xi\zeta}s_{\zeta\zeta}^{-1}s_{\xi\zeta}'\right)^{-1}s_{\xi\zeta}s_{\zeta\zeta}^{-1}$ can be consistently estimated from the observed data, ψ_{2SLS} cannot be estimated without access to the *true* values of the AI-coded variables (at least in a calibration subsample). Without an estimate of ψ_{2SLS}, the direction and magnitude of the bias remain unknown.

This ambiguity has critical implications for hypothesis testing. It implies that AI-coding opens the door to both Type I and Type II errors—and thus to the potential for both rejecting accurate theories and favoring the development of inaccurate theories—as statistical tests of significance critically depend on estimated effect size. While a smaller than true estimated effect size is likely to lead to a Type II error, a larger than true effect size is likely to lead to a Type I error. This underscores the importance of correcting for AI-coding bias to ensure valid statistical inference and theory development.

C.2 Bias Correction

We build on the method of attenuation correction in OLS, which adjusts for measurement error by scaling the estimated coefficient using the reliability ratio—the ratio of the variance of the true variable to the variance of the observed (AI-coded) variable (Fuller 2009):

$$\operatorname*{plim}_{N\to\infty} d_{\text{2SLS}} = \underbrace{\frac{\sigma_\xi^2}{\sigma_\xi^2+\sigma_\eta^2}}_{\text{Reliability Ratio}}\delta. \tag{26}$$

where σ_ξ^2 is the variance of the true independent variable, and σ_η^2 is the variance of the AI-coding error. While this approach is useful in simple OLS settings, it does not apply to multivariate

regression or account for endogeneity in instrumental variable (IV) regression. It also does not extend to cases where classical error assumptions are relaxed, as discussed in the prior subsection. We therefore develop a more general correction method based on the bias structure derived in Equation (23).

From Equation (23), the probability limit of the naïve 2SLS estimator is:

$$\operatorname*{plim}_{N\to\infty} d_{\text{2SLS}} = \delta \; - \; \underbrace{\left[s_{\xi\zeta}\, s_{\zeta\zeta}^{-1}\, s_{\xi\zeta}'\right]^{-1} s_{\xi\zeta}\, s_{\zeta\zeta}^{-1}\, \psi_{\text{2SLS}}\, \delta}_{\text{Bias Term}}, \tag{27}$$

where $s_{\xi\zeta}$ and $s_{\zeta\zeta}$ are probability-limit cross-product matrices of regressors and instruments, and ψ_{2SLS} is the probability-limit cross-product of the instruments with the AI-coding errors. Rearranging, we obtain:

$$d_{\text{2SLS}} = (I_g - \text{BiasFactor}_{\text{2SLS}})\, \delta, \tag{28}$$

where:

$$\text{BiasFactor}_{\text{2SLS}} = \left[s_{\xi\zeta}\, s_{\zeta\zeta}^{-1}\, s_{\xi\zeta}'\right]^{-1} s_{\xi\zeta}\, s_{\zeta\zeta}^{-1}\, \psi_{\text{2SLS}}. \tag{29}$$

Since $\text{BiasFactor}_{\text{2SLS}}$ is generally nonzero, the naïve 2SLS estimator is biased. To correct for this bias, we solve for δ by inverting $(I_g - \text{BiasFactor}_{\text{2SLS}})$ and premultiplying both sides of Equation (28) by $(I_g - \text{BiasFactor}_{\text{2SLS}})^{-1}$:

$$\delta = (I_g - \text{BiasFactor}_{\text{2SLS}})^{-1} d_{\text{2SLS}}. \tag{30}$$

In practice, we replace the population-level quantities $\left(s_{\xi\zeta}, s_{\zeta\zeta}, \psi_{\text{2SLS}}\right)$ with their consistent sample estimates $\left(\hat{s}_{\xi\zeta}, \hat{s}_{\zeta\zeta}, \hat{\psi}_{\text{2SLS}}\right)$. Define the estimated bias factor:

$$\widehat{\text{BiasFactor}}_{\text{2SLS}} = \left[\hat{s}_{\xi\zeta}\, \hat{s}_{\zeta\zeta}^{-1}\, \hat{s}_{\xi\zeta}'\right]^{-1} \hat{s}_{\xi\zeta}\, \hat{s}_{\zeta\zeta}^{-1}\, \hat{\psi}_{\text{2SLS}}. \tag{31}$$

Then, the corrected estimator $d_{\text{2SLS}}^{\text{con}}$ is obtained by premultiplying the naive estimator, d_{2SLS},

by the inverse of $(I_g - \widehat{\text{BiasFactor}}_{2SLS})$:

$$d_{2SLS}^{con} = (I_g - \widehat{\text{BiasFactor}}_{2SLS})^{-1} d_{2SLS}. \tag{32}$$

We define $\text{CorrectionMatrix}_{2SLS} = (I_g - \widehat{\text{BiasFactor}}_{2SLS})^{-1}$, so that:

$$d_{2SLS}^{con} = \text{CorrectionMatrix}_{2SLS}\, d_{2SLS}. \tag{33}$$

By Slutsky's theorem, since $\hat{s}_{\xi\xi}$, $\hat{s}_{\zeta\zeta}$, and $\hat{\psi}_{2SLS}$ are consistent estimators of $s_{\xi\xi}$, $s_{\zeta\zeta}$, and ψ_{2SLS}, respectively, $\widehat{\text{BiasFactor}}_{2SLS}$ converges in probability to BiasFactor_{2SLS}. Also, from Equation (27), d_{2SLS} converges in probability to $(I_g - \text{BiasFactor}_{2SLS})\delta$. Since matrix inversion is a continuous function, the continuous mapping theorem implies that d_{2SLS}^{con} converges in probability to the solution of Equation (30):

$$\operatorname*{plim}_{N\to\infty} d_{2SLS}^{con} = \delta. \tag{34}$$

This derivation confirms that the matrix-inverse correction effectively removes the bias caused by AI-coding errors in 2SLS estimation. The key insight is that the difference $d_{2SLS} - \delta$ is known in closed form as a function of δ (and estimable quantities), allowing us to solve for δ explicitly.

C.3 Asymptotic Normality and Inference

Having established that our bias-corrected estimator is consistent, we now demonstrate that it is also asymptotically normal and, hence, suitable for standard inference procedures. Recall from the correction derivation that the BiasFactor_{2SLS} is defined by

$$\text{BiasFactor}_{2SLS} = A_{2SLS}^{-1} B_{2SLS}, \tag{35}$$

where

$$A_{2SLS} = s_{\xi\zeta}\, s_{\zeta\zeta}^{-1}\, s_{\xi\zeta}', \quad B_{2SLS} = s_{\xi\zeta}\, s_{\zeta\zeta}^{-1}\, \psi_{2SLS}. \tag{36}$$

Thus, we can write the probability limit of the naïve estimator as

$$\operatorname*{plim}_{N\to\infty} d_{\text{2SLS}} = \left(I_g - A_{\text{2SLS}}^{-1} B_{\text{2SLS}}\right)\delta. \tag{37}$$

Moreover, the standard (naïve) 2SLS estimator satisfies

$$\sqrt{N}\left[d_{\text{2SLS}} - \left(I_g - A_{\text{2SLS}}^{-1} B_{\text{2SLS}}\right)\delta\right] \xrightarrow{d} \mathcal{N}(0,\, V_{\text{2SLS}}), \tag{38}$$

where V_{2SLS} denotes the asymptotic variance of the naïve estimator.

Since the correction function

$$h_{\text{2SLS}}(d_{\text{2SLS}}) = \left(I_g - \widehat{\text{BiasFactor}}_{\text{2SLS}}\right)^{-1} d_{\text{2SLS}} \tag{39}$$

is continuously differentiable in a neighborhood of the probability limit $\left(I_g - A_{\text{2SLS}}^{-1} B_{\text{2SLS}}\right)\delta$ and because $\widehat{\text{BiasFactor}}_{\text{2SLS}}$ converges in probability to $A_{\text{2SLS}}^{-1} B_{\text{2SLS}}$, the continuous mapping theorem and the delta method imply that

$$\sqrt{N}\left(d_{\text{2SLS}}^{\text{con}} - \delta\right) = \sqrt{N}\left[\left(I_g - \widehat{\text{BiasFactor}}_{\text{2SLS}}\right)^{-1} d_{\text{2SLS}} \;-\; \delta\right] \tag{40}$$

$$\xrightarrow{d} \mathcal{N}\Big(0,\; \left(I_g - A_{\text{2SLS}}^{-1} B_{\text{2SLS}}\right)^{-1} V_{\text{2SLS}} \left[\left(I_g - A_{\text{2SLS}}^{-1} B_{\text{2SLS}}\right)^{-1}\right]'\Big). \tag{41}$$

In other words, although the naïve estimator d_{2SLS} is biased due to AI-coding errors, our correction—by premultiplying with $(I_g - \widehat{\text{BiasFactor}}_{\text{2SLS}})^{-1}$—removes that bias and, via a first-order Taylor expansion of $h(d)$, preserves asymptotic normality with a transformed asymptotic covariance matrix. This result justifies the use of conventional statistical inference tools—such as hypothesis testing and the construction of confidence intervals—either by using the delta method or, as we recommend in practice, via bootstrap methods that consistently approximate the distribution of $d_{\text{2SLS}}^{\text{con}}$.

C.4 Identification

The presence of AI-coding errors introduces additional identification requirements beyond those of standard 2SLS. Specifically, for the corrected estimator $d_{\text{2SLS}}^{\text{con}}$ to be well-defined, the matrices

$$s_{\xi\xi} = \operatorname*{plim}_{N\to\infty} \frac{\hat{\xi}'\hat{\xi}}{N} \quad \text{and} \quad s_{\zeta\zeta} = \operatorname*{plim}_{N\to\infty} \frac{\hat{\zeta}'\hat{\zeta}}{N} \tag{42}$$

must have full column rank as $N \to \infty$. This follows from Equation (31): if these matrices do not have full rank, the corresponding product matrix $\widehat{\text{BiasFactor}}_{\text{2SLS}}$ would be singular, making its inverse—and hence the bias correction—ill-defined. This requirement mirrors the standard 2SLS identification conditions, which require that

$$\operatorname*{plim}_{N\to\infty} \frac{\xi'\zeta}{N} \quad \text{and} \quad \operatorname*{plim}_{N\to\infty} \frac{\zeta'\zeta}{N} \tag{43}$$

have full column rank. However, a key difference is that correlations between AI-coding errors (η_ξ, η_ζ) and the regressors or instruments can reduce the rank of the *observed* cross-product matrices $s_{\xi\zeta}$ and $s_{\zeta\zeta}$, even when the corresponding *true* cross-product matrices remain full rank. As a result, the model may become underidentified due to these correlations, despite satisfying classical identification conditions.

Moreover, identification requires that the matrix $\text{CorrectionMatrix}_{\text{2SLS}}$ be invertible. Recall that:

$$\text{CorrectionMatrix}_{\text{2SLS}} = \left(I_g - \left(\hat{s}_{\xi\zeta} \hat{s}_{\zeta\zeta}^{-1} \hat{s}_{\xi\zeta}' \right)^{-1} \hat{s}_{\xi\zeta} \hat{s}_{\zeta\zeta}^{-1} \hat{\psi}_{\text{2SLS}} \right)^{-1}. \tag{44}$$

This condition is satisfied if the true regressors and true instruments are sufficiently correlated. To see this, consider the condition for $\text{CorrectionMatrix}_{\text{2SLS}}$ to be invertible, which is equivalent to requiring that $(I_g - \widehat{\text{BiasFactor}}_{\text{2SLS}})$ be invertible, which in turn is equivalent to:

$$\operatorname*{plim}_{N\to\infty} \frac{\hat{\xi}'\hat{\zeta}}{N} \neq \operatorname*{plim}_{N\to\infty} \frac{\eta_\xi'\hat{\zeta}}{N}. \tag{45}$$

Since $\hat{\xi} = \xi + \eta_\xi$ and $\hat{\zeta} = \zeta + \eta_\zeta$, we can rewrite this as:

$$s_{\xi\zeta} + \psi'_{\text{2SLS}} \neq \psi'_{\text{2SLS}}. \tag{46}$$

This simplifies to the requirement that:

$$s_{\xi\zeta} \neq 0. \tag{47}$$

Equation (47) states that the *true* regressors and *true* instruments must be correlated in the limit. If $s_{\xi\zeta} = 0$, the model is unidentified even *without* AI-coding error. The AI-coding error makes identification *more difficult* by potentially reducing the rank of the observed cross-product matrices, but it doesn't change the fundamental requirement of a relationship between the true regressors and instruments, as is typical in IV and 2SLS estimation.

In practice, the extent to which these identification conditions are satisfied can be assessed by examining the condition numbers of $\hat{s}_{\xi\zeta}$, $\hat{s}_{\zeta\zeta}$, and $\text{CorrectionMatrix}_{\text{2SLS}}$, all of which are computed from the calibration subsample. A *relatively* low condition number indicates strong instruments and a well-posed model, while a high condition number suggests a lack of identification and potential numerical instability in the estimator.

C.5 Algorithm

We propose estimating $s_{\xi\zeta}$, $s_{\zeta\zeta}$, and $s_{\zeta y}$ in the full sample, while estimating ψ_{2SLS} in the manually-coded calibration subsample where both AI-coded and precisely coded values are available. Estimating parameters in the full sample improves precision, whereas estimation in the smaller calibration subsample sacrifices some precision. While it is possible to estimate all parameters within the smaller subsample, doing so would result in a significant loss of statistical power. Conversely, estimating all parameters using the full sample is impractical because developing precisely coded variables for the entire sample would be cost-prohibitive. Thus, our proposed estimation framework seeks to balance efficiency and accuracy by leveraging the full sample where feasible and using the calibration subsample for parameters that require precisely coded

data. Since $d_{2\text{SLS}}^{\text{con}}$ does not have closed-form standard errors, we propose bootstrapping. Algorithm 6 outlines the complete process, where nB is the number of bootstrap iterations, Data_s denotes the manually coded calibration subsample, and Data_f the full sample. N_s and N_f are the number of draws from Data_s and Data_f respectively.

Algorithm 6 Algorithm for Inference in Single Equation Marketing Models

Input: Manually coded calibration subsample (Data_s), AI-coded full sample (Data_f).

▷ Steps 1-3 (data acquisition, manual coding, AI coding) are assumed to be complete.

for $b \leftarrow 1, nB$ **do**

Draw bootstrap sample Bootstrap Calibration Subsample$[b]$ of size N_s from Data_s *with replacement*.

Infer $\hat{\psi}_{2\text{SLS}} = \frac{\hat{\zeta}'\eta_\xi}{N_s}$ in Bootstrap Calibration Subsample$[b]$.

Draw bootstrap sample Bootstrap Full Sample$[b]$ of size N_f from Data_f *with replacement*.

Compute the canonical 2SLS estimator, $d_{2\text{SLS}}[b] = \left[\hat{\xi}'\hat{\zeta}\left(\hat{\zeta}'\hat{\zeta}\right)^{-1}\hat{\zeta}'\hat{\xi}\right]^{-1}\hat{\xi}'\hat{\zeta}\left(\hat{\zeta}'\hat{\zeta}\right)^{-1}\hat{\zeta}'y$, in Bootstrap Full Sample$[b]$.

Infer $\hat{s}_{\xi\zeta} = \frac{\hat{\xi}'\hat{\zeta}}{N_f}$ and $\hat{s}_{\zeta\zeta} = \frac{\hat{\zeta}'\hat{\zeta}}{N_f}$ in Bootstrap Full Sample$[b]$.

Calculate $\widehat{\text{BiasFactor}}_{2\text{SLS}}[b] = \left[\hat{s}_{\xi\zeta}\,\hat{s}_{\zeta\zeta}^{-1}\,\hat{s}_{\xi\zeta}'\right]^{-1}\hat{s}_{\xi\zeta}\,\hat{s}_{\zeta\zeta}^{-1}\,\hat{\psi}_{2\text{SLS}}$.

$d_{2\text{SLS}}^{con}[b] = (I_g - \widehat{\text{BiasFactor}}_{2\text{SLS}}[b])^{-1}d_{2\text{SLS}}[b]$.

end for

Infer test statistics from the distribution of $d_{2\text{SLS}}^{con}[b], \quad b = 1, \ldots, nB$.

Crucially, $\psi_{2\text{SLS}}$ is likely to be sparse because it represents the cross-product of the instruments $\hat{\zeta}$ and the coding errors η_ξ. Specifically, $\psi_{2\text{SLS}}$ is a matrix of size $h \times g$, where h is the number of instruments and g is the number of regressors. Columns of $\psi_{2\text{SLS}}$ corresponding to regressors that are observed and not coded with error are zero. Therefore, the number of non-zero elements in $\psi_{2\text{SLS}}$ is at most the product of the number of AI-coded variables and the number of instruments h; all other elements are zero. This sparsity reduces the variance of the estimator of $\psi_{2\text{SLS}}$, making the calibration subsample approach more feasible, since fewer parameters need to be estimated from the smaller subsample. In contrast, $s_{\xi\zeta}$, $s_{\zeta\zeta}$, and $s_{\zeta y}$ are generally dense matrices with many non-zero elements, making them better suited to estimation using the full sample.

Statistical testing is conducted in Algorithm 6 by computing the corresponding percentile values of the bootstrap estimates (e.g., the 2.5th and 97.5th percentiles for a two-tailed 95%

confidence interval) and then comparing the null hypothesis value to this interval. If the null value falls within the confidence interval, then the null hypothesis cannot be rejected at the corresponding significance level; if the null value falls outside the confidence interval, the null hypothesis is rejected.

D Addressing AI-Coding Errors in System-of-Equations Marketing Models

System-of-equations models are frequently employed in marketing research to analyze scenarios where multiple, related outcomes are influenced by a common set of factors. These models are particularly relevant when the structural errors across equations are correlated, reflecting unobserved variables or shared shocks that affect multiple outcomes simultaneously. Indeed, as demonstrated in the main paper's simulation studies and observed in many real-world marketing contexts, such interdependencies are common.

Extending the single-equation example from Web Appendix §C, consider a study examining how humor in social media posts affects both user engagement (likes) and user interaction (comments). We might hypothesize that humor influences both outcomes, but perhaps in different ways or to different degrees. Furthermore, there might be unobserved factors (e.g., the overall quality of the post, the poster's reputation, or a trending topic) that influence both likes and comments, leading to correlated errors across the equations.

Analyzing these outcomes jointly within a system-of-equations framework, rather than individually, offers several advantages. First, system estimators (like three-stage least squares, 3SLS) can be more efficient than single-equation estimators (like two-stage least squares, 2SLS) by exploiting the information contained in the cross-equation error correlations. Second, system-of-equations models allow for testing cross-equation restrictions, enabling researchers to examine hypotheses that span multiple relationships (e.g., "Does humor have a stronger effect on likes than on comments?").

However, the introduction of AI-coded variables into system-of-equations models presents unique challenges. As shown in the single-equation context, AI-coding errors are often non-classical, potentially correlated with other regressors or instruments. In a system-of-equations setting, these errors can propagate *across equations*, compounding the biases and potentially leading to even more misleading inferences. For example, if the AI model used to code humor

is more accurate for certain types of posts (e.g., those with clear visual gags) than others (e.g., those with subtle wordplay), and if those post types also differ systematically in their engagement and interaction rates, then the AI-coding error for humor will be correlated with other factors influencing both likes and comments. This correlation will not only bias the humor coefficient in *both* equations but will also contaminate the estimates of *other* coefficients in both equations due to the shared, correlated error structure. This cross-equation contamination is a critical issue that must be addressed to ensure valid inference.

This section extends the bias-correction framework developed for single-equation models in the prior section to the system-of-equations context. We derive the bias in the canonical 3SLS estimator when AI-coding errors are present, demonstrating how these errors propagate across equations. We then propose a consistent estimator that accounts for these complex, cross-equation biases, building upon the insights and methodology established in the single-equation case. Finally, we provide a detailed algorithm for implementing the corrected 3SLS estimator, including the crucial bootstrap procedure for obtaining valid standard errors.

D.1 Setup

To formalize the system-of-equations framework, we extend our prior example of a researcher analyzing how humor in social media posts affects two outcomes: likes (y_1) and comments (y_2). The structural relationships are given by:

$$\begin{aligned} \text{likes}_i &= \alpha_1 + \beta_1 \cdot \text{humor}_i + \gamma_1 \cdot \text{post\_time}_i + \varepsilon_{1i}, \\ \text{comments}_i &= \alpha_2 + \beta_2 \cdot \text{humor}_i + \gamma_2 \cdot \text{question}_i + \varepsilon_{2i}, \end{aligned} \tag{48}$$

where humor_i is a latent variable representing the humor intensity of the post, which is AI-coded as $\hat{\text{humor}}_i = \text{humor}_i + \eta_i$. The variable post_time_i captures the time of day the post was made, and question_i indicates the presence of a question in the post; both are observed without error. The time of day is hypothesized to influence likes, as posts made at certain times may reach a larger audience. Conversely, the presence of a question is hypothesized to influence comments, as

it may encourage interaction. The error terms $(\varepsilon_{1i}, \varepsilon_{2i})$ are correlated due to shared unobservables, such as the overall quality of the post or its relevance to current trends.

More generally, marketing researchers may face a system of G equations, where each equation g corresponds to a distinct dependent variable y_g (e.g., likes, comments) and a set of regressors ξ_g. These regressors may include both shared covariates (e.g., humor intensity) and equation-specific variables (e.g., post timing, presence of a question). Each equation takes the general form introduced in Equation (19); the single-equation model is nested within the system-of-equations model:

$$y_{gi} = \alpha_g + \sum_{m=1}^{M_g} \beta_{gm} z_{gmi} + \sum_{l=1}^{L_g} \gamma_{gl} x_{gli} + \varepsilon_{gi}, \tag{49}$$

where z_{gmi} represents AI-coded variables (e.g., humor), x_{gli} observed variables (e.g., post timing, presence of a question), and ε_{gi} errors that may correlate across equations due to shared unobservables (e.g., viral trends, overall post quality).

Let $Y = \text{vec}(y_1, \ldots, y_G)$ stack all dependent variables, $\Xi = \text{diag}(\xi_1, \ldots, \xi_G)$ represent the block-diagonal matrix of regressors, and $\Delta = \text{vec}(\delta_1, \ldots, \delta_G)$ collect all coefficients. The error structure is represented by $E = \text{vec}(\varepsilon_1, \ldots, \varepsilon_G)$, with $\text{Var}(E) = \Sigma \otimes I_N$, allowing for correlated shocks across equations. Here, Σ is the covariance matrix of contemporaneous errors, I_N is an $N \times N$ identity matrix, and $\otimes$ denotes the Kronecker product. Then the system of equations can be depicted as follows:

$$\underbrace{\begin{bmatrix} y_1 \\ y_2 \\ \ldots \\ y_G \end{bmatrix}}_{Y} = \underbrace{\begin{bmatrix} \xi_1 & 0 & 0 & \ldots \\ 0 & \xi_2 & 0 & \ldots \\ \vdots & \vdots & \ddots & \\ 0 & 0 & & \xi_G \end{bmatrix}}_{\Xi} \underbrace{\begin{bmatrix} \delta_1 \\ \delta_2 \\ \ldots \\ \delta_G \end{bmatrix}}_{\Delta} + \underbrace{\begin{bmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \vdots \\ \varepsilon_G \end{bmatrix}}_{E} \tag{50}$$

The variables that are coded with error are denoted as follows:

$$\underbrace{\begin{bmatrix} \hat{\xi}_1 & 0 & 0 & \dots \\ 0 & \hat{\xi}_2 & 0 & \dots \\ \vdots & \vdots & \ddots & \\ 0 & 0 & & \hat{\xi}_G \end{bmatrix}}_{\hat{\Xi}} = \underbrace{\begin{bmatrix} \xi_1 & 0 & 0 & \dots \\ 0 & \xi_2 & 0 & \dots \\ \vdots & \vdots & \ddots & \\ 0 & 0 & & \xi_G \end{bmatrix}}_{\Xi} + \underbrace{\begin{bmatrix} \eta_{\xi 1} & 0 & 0 & \dots \\ 0 & \eta_{\xi 2} & 0 & \dots \\ \vdots & \vdots & \ddots & \\ 0 & 0 & & \eta_{\xi G} \end{bmatrix}}_{H_\xi}, \tag{51}$$

$$\underbrace{\begin{bmatrix} \hat{\zeta}_1 & 0 & 0 & \dots \\ 0 & \hat{\zeta}_2 & 0 & \dots \\ \vdots & \vdots & \ddots & \\ 0 & 0 & & \hat{\zeta}_G \end{bmatrix}}_{\vec{\hat{\zeta}}} = \underbrace{\begin{bmatrix} \zeta_1 & 0 & 0 & \dots \\ 0 & \zeta_2 & 0 & \dots \\ \vdots & \vdots & \ddots & \\ 0 & 0 & & \zeta_G \end{bmatrix}}_{\vec{\zeta}} + \underbrace{\begin{bmatrix} \eta_{\zeta 1} & 0 & 0 & \dots \\ 0 & \eta_{\zeta 2} & 0 & \dots \\ \vdots & \vdots & \ddots & \\ 0 & 0 & & \eta_{\zeta G} \end{bmatrix}}_{H_\zeta}, \tag{52}$$

where $\hat{}$ denotes the coded or observed values of a variable. Equation (51) denotes the observation model for the regressors, and Equation (52) denotes the observation model for the instruments.

D.2 Bias Direction

We begin with the case $\zeta \equiv \zeta_1 = \zeta_2 = \cdots = \zeta_G$, meaning that the same set of instruments is used across all equations. This scenario is particularly relevant in marketing applications where multiple interrelated outcomes—such as likes and comments on social media posts—may be influenced by common factors. For instance, in our prior example, both likes and comments on a post may be functions of humor intensity (which is AI-coded) and other observed features like post timing or the presence of a question. If humor intensity is endogenous, instruments such as lagged engagement metrics or external events (e.g., a comedy festival) may be used to correct for endogeneity, and these instruments would typically be shared across equations.

Similar to our focus on the 2SLS estimator in single-equation models, we focus on the 3SLS

estimator:

$$D_{3\text{SLS}} = \left[\hat{\Xi}'\left(I_G \otimes \vec{\hat{\zeta}}\right)\left(\hat{\Sigma} \otimes \vec{\hat{\zeta}}'\vec{\hat{\zeta}}\right)^{-1}\left(I_G \otimes \vec{\hat{\zeta}}'\right)\hat{\Xi}\right]^{-1}\hat{\Xi}'\left(I_G \otimes \vec{\hat{\zeta}}\right)\left(\hat{\Sigma} \otimes \vec{\hat{\zeta}}'\vec{\hat{\zeta}}\right)^{-1}\left(I_G \otimes \vec{\hat{\zeta}}'\right)Y, \tag{53}$$

$$= \left[\hat{\Xi}'\left(\hat{\Sigma}^{-1} \otimes \overline{P}_{3\text{SLS}}\right)\hat{\Xi}\right]^{-1}\hat{\Xi}'\left(\hat{\Sigma}^{-1} \otimes \overline{P}_{3\text{SLS}}\right)Y, \tag{54}$$

where $\overline{P}_{3\text{SLS}} = \vec{\hat{\zeta}}\left(\vec{\hat{\zeta}}'\vec{\hat{\zeta}}\right)^{-1}\vec{\hat{\zeta}}'$.[11] Here, $\hat{\Xi}$ and $\vec{\hat{\zeta}}$ are as defined in Equations (51) and (52) respectively, and $\hat{\Sigma}$ is a consistent estimator for Σ. Moreover, for each equation g we have

$$y_g = \left(\hat{\xi}_g - \eta_{\xi g}\right)\delta_g + \varepsilon_g = \hat{\xi}_g\delta_g + u_g, \quad g = 1, \ldots, G, \tag{55}$$

where $u_g = \varepsilon_g - \eta_{\xi g}\delta_g$ and $U = \text{vec}(u_1, \ldots, u_G)$.

The asymptotics of the 3SLS estimator are given by:

$$\operatorname*{plim}_{N\to\infty} D_{3\text{SLS}} = \Delta + \operatorname*{plim}_{N\to\infty}\left[\hat{\Xi}'\left(\hat{\Sigma}^{-1} \otimes \overline{P}_{3\text{SLS}}\right)\hat{\Xi}\right]^{-1}\hat{\Xi}'\left(\hat{\Sigma}^{-1} \otimes \overline{P}_{3\text{SLS}}\right)U, \tag{56}$$

$$= \Delta - \underbrace{\operatorname*{plim}_{N\to\infty}\left[\hat{\Xi}'\left(\hat{\Sigma}^{-1} \otimes \overline{P}_{3\text{SLS}}\right)\hat{\Xi}\right]^{-1}\hat{\Xi}'\left(\hat{\Sigma}^{-1} \otimes \overline{P}_{3\text{SLS}}\right)\left(H_\xi\Delta\right)}_{\text{Bias}}, \tag{57}$$

where the simplification in the last step follows from analogous arguments as in Equation (23): the structural errors are orthogonal to the true values of the instruments ($\text{plim}_{N\to\infty}\frac{1}{N}\zeta'E = 0$), and to the coding errors in the instruments ($\text{plim}_{N\to\infty}\frac{1}{N}\eta_\zeta'E = 0$), which in part follows as the instruments (by definition) are variables that are orthogonal to the structural errors.

Let $\Delta[i]$ denote the i-th element of the true coefficient vector Δ. For instance, because Δ is a stacked coefficient vector, for $i \leq g$ the element $\Delta[i]$ is the i-th coefficient of the first equation. From Equation (57), the asymptotic bias of the i-th coefficient in the uncorrected 3SLS estimator

[11] Given matrices A, B, C, and D such that AC and BD are well defined, $(A \otimes B)(C \otimes D) = AC \otimes BD$. If A and B are nonsingular then $(A \otimes B)^{-1} = A^{-1} \otimes B^{-1}$.

is given by:

$$\text{Bias}\left(\operatorname*{plim}_{N\to\infty} D_{3\text{SLS}}[i]\right) = -\operatorname*{plim}_{N\to\infty}\left[\left(\hat{\Xi}'\left(\hat{\Sigma}^{-1}\otimes\overline{P}_{3\text{SLS}}\right)\hat{\Xi}\right)^{-1}\hat{\Xi}'\left(\hat{\Sigma}^{-1}\otimes\overline{P}_{3\text{SLS}}\right)\left(H_{\xi}\Delta\right)\right][i], \tag{58}$$

where $A[i]$ denotes the i-th element of the vector A.

Equation (58) nests Equation (24). To see this, define the probability limits of the moment matrices:

$$A_{3\text{SLS}} = \operatorname*{plim}_{N\to\infty}\frac{\hat{\Xi}'\left(\hat{\Sigma}^{-1}\otimes\overline{P}_{3\text{SLS}}\right)\hat{\Xi}}{N} \quad \text{and} \quad B_{3\text{SLS}} = \operatorname*{plim}_{N\to\infty}\frac{\hat{\Xi}'\left(\hat{\Sigma}^{-1}\otimes\overline{P}_{3\text{SLS}}\right)H_{\xi}}{N}. \tag{59}$$

Then, the asymptotic probability limit of the 3SLS estimator is given by

$$\operatorname*{plim}_{N\to\infty} D_{3\text{SLS}}[i] \;=\; \Delta[i] \;-\; A_{3\text{SLS}}^{-1}B_{3\text{SLS}}\Delta[i], \tag{60}$$

which, in the case of a single equation and Σ equal to the identity matrix, resolves to the expression in Equation (24).

These results imply that the findings from the single equation model and 2SLS extend to the 3SLS estimator; namely, AI-coding error in a regressor can bias (both attenuate and amplify) the estimates of the coefficients associated with that regressor, and such bias can proliferate to the coefficients of other regressors that are observed (and therefore not suffer from coding error). In addition, moreover, Equation (58) shows that coding error in a variable that is only included in one equation can propagate to the coefficients in other equations. This is because, in the definitions of $A_{3\text{SLS}}$ and $B_{3\text{SLS}}$, $\hat{\Sigma}^{-1}\otimes\overline{P}_{3\text{SLS}}$ aggregates information from all equations. These issues caution against the naïve inclusion of AI-coded variables in system-of-equations models.

D.3 Bias Correction

To address these issues, we extend the bias-correction framework developed for single-equation models to the system-of-equations context. Just as OLS with classical measurement error can be

corrected using a reliability ratio, and 2SLS can be adjusted using a matrix generalization of this concept, our approach generalizes this principle to 3SLS. To facilitate implementation, we derive a more tractable expression for the bias in terms of cross-covariance matrices in Equation (61). This expression forms the basis for our proposed estimator, introduced in Equation (64). We begin with the base case where the same instruments apply to all equations.[12]

$$\operatorname*{plim}_{N\to\infty} D_{\text{3SLS}} = \Delta - \underbrace{\begin{bmatrix} \Sigma_{11}^{-1} s_{\xi 1\zeta} s_{\zeta\zeta}^{-1} s_{\xi 1\zeta}' & \Sigma_{12}^{-1} s_{\xi 1\zeta} s_{\zeta\zeta}^{-1} s_{\xi 2\zeta}' & \cdots \\ \Sigma_{21}^{-1} s_{\xi 2\zeta} s_{\zeta\zeta}^{-1} s_{\xi 1\zeta}' & \Sigma_{22}^{-1} s_{\xi 2\zeta} s_{\zeta\zeta}^{-1} s_{\xi 2\zeta}' & \cdots \\ \vdots & \ddots & \\ \cdots & \cdots & \Sigma_{GG}^{-1} s_{\xi G\zeta} s_{\zeta\zeta}^{-1} s_{\xi G\zeta}' \end{bmatrix}^{-1}}_{:=A_{\text{3SLS}}^{-1}} \underbrace{\begin{bmatrix} \Sigma_{11}^{-1} s_{\xi 1\zeta} s_{\zeta\zeta}^{-1} \psi_1 & \Sigma_{12}^{-1} s_{\xi 1\zeta} s_{\zeta\zeta}^{-1} \psi_2 & \cdots \\ \Sigma_{21}^{-1} s_{\xi 2\zeta} s_{\zeta\zeta}^{-1} \psi_1 & \Sigma_{22}^{-1} s_{\xi 2\zeta} s_{\zeta\zeta}^{-1} \psi_2 & \cdots \\ \vdots & \ddots & \\ \cdots & \cdots & \Sigma_{GG}^{-1} s_{\xi G\zeta} s_{\zeta\zeta}^{-1} \psi_G \end{bmatrix}}_{:=B_{\text{3SLS}}} \Delta, \quad (61)$$

where ψ_g is the $h \times g$ cross-covariance matrix, defined as $\operatorname{plim}_{N\to\infty} \frac{\hat{\zeta}' \eta_{\xi i}}{N}$. The terms $s_{\zeta\zeta} = \operatorname{plim}_{N\to\infty} \frac{\hat{\zeta}'\hat{\zeta}}{N}$ and $s_{\xi i\zeta} = \operatorname{plim}_{N\to\infty} \frac{\hat{\xi}_i'\hat{\zeta}}{N}$ represent the probability limits of the sample cross-product matrices of the observed instruments and regressors, respectively. The matrices A_{3SLS} and B_{3SLS} are of dimension $gG \times gG$.

Let I_{Gg} be the $Gg \times Gg$ identity matrix. Then, we can rewrite Equation (61) as:

$$D_{\text{3SLS}} = (I_{Gg} - A_{\text{3SLS}}^{-1} B_{\text{3SLS}})\Delta. \quad (62)$$

This equation is analogous to Equation (28) in the single-equation case. It highlights the structural similarity of the 3SLS bias to the 2SLS bias: the naive 3SLS estimator is a linear projection of the

[12] It is important to note that different equations can incorporate distinct regressors.

true parameters, Δ.

Let $\hat{A}_{3SLS}$ and $\hat{B}_{3SLS}$ be consistent estimators of A_{3SLS} and B_{3SLS}, respectively. These estimators are obtained by replacing the population quantities in Equation (61) with their sample counterparts—using the calibration subsample for terms involving ψ_g and the full sample for all other terms. We define the estimated bias factor $\widehat{\text{BiasFactor}}_{3SLS}$ for the base case as:

$$\widehat{\text{BiasFactor}}_{3SLS} = \hat{A}_{3SLS}^{-1}\hat{B}_{3SLS}. \tag{63}$$

Our proposed consistent estimator, D_{3SLS}^{con}, is then obtained by solving the following linear system:

$$\left[I_{Gg} - \widehat{\text{BiasFactor}}_{3SLS}\right] D_{3SLS}^{con} = D_{3SLS}. \tag{64}$$

Or, equivalently:

$$D_{3SLS}^{con} = \left[I_{Gg} - \widehat{\text{BiasFactor}}_{3SLS}\right]^{-1} D_{3SLS}. \tag{65}$$

This is the direct analog of Equation (32) in the single-equation case. We are premultiplying the naive 3SLS estimator D_{3SLS} by the inverse of $(I_{Gg} - \widehat{\text{BiasFactor}}_{3SLS})$ to correct for the bias. Therefore, by Slutsky's theorem and the continuous mapping theorem (similar to the single-equation case), since $\hat{A}_{3SLS}$ and $\hat{B}_{3SLS}$ are consistent estimators, and since D_{3SLS} converges in probability to $(I_{Gg} - A_{3SLS}^{-1}B_{3SLS})\Delta$, it follows that D_{3SLS}^{con} converges in probability to Δ:

$$\operatorname*{plim}_{N\to\infty} D_{3SLS}^{con} = \Delta. \tag{66}$$

In the more complex scenario where different equations have different instruments, we cannot represent the 3SLS estimator using Kronecker products. Instead, the 3SLS estimator is represented

as:

$$D_{\text{3SLS}} = \left[\hat{\Xi}'\vec{\hat{\zeta}}\begin{bmatrix}\hat{\Sigma}_{11}\hat{\zeta}_1'\hat{\zeta}_1 & \hat{\Sigma}_{12}\hat{\zeta}_1'\hat{\zeta}_2 & \dots \\ \hat{\Sigma}_{21}\hat{\zeta}_2'\hat{\zeta}_1 & \hat{\Sigma}_{22}\hat{\zeta}_2'\hat{\zeta}_2 & \dots \\ \vdots & \ddots & \\ \dots & \dots & \hat{\Sigma}_{GG}\hat{\zeta}_G'\hat{\zeta}_G\end{bmatrix}^{-1}\vec{\hat{\zeta}}'\hat{\Xi}\right]^{-1}\hat{\Xi}'\vec{\hat{\zeta}}\begin{bmatrix}\hat{\Sigma}_{11}\hat{\zeta}_1'\hat{\zeta}_1 & \hat{\Sigma}_{12}\hat{\zeta}_1'\hat{\zeta}_2 & \dots \\ \hat{\Sigma}_{21}\hat{\zeta}_2'\hat{\zeta}_1 & \hat{\Sigma}_{22}\hat{\zeta}_2'\hat{\zeta}_2 & \dots \\ \vdots & \ddots & \\ \dots & \dots & \hat{\Sigma}_{GG}\hat{\zeta}_G'\hat{\zeta}_G\end{bmatrix}^{-1}\vec{\hat{\zeta}}'Y. \tag{67}$$

In this scenario, the asymptotics of the 3SLS estimator can be described by:

$$\begin{aligned}\operatorname*{plim}_{N\to\infty} D_{\text{3SLS}} = \Delta - \underbrace{\left[\begin{bmatrix}s_{\xi 1\zeta 1} & 0 & \dots \\ 0 & s_{\xi 2\zeta 2} & \dots \\ \vdots & \ddots & \\ \dots & \dots & s_{\xi G\zeta G}\end{bmatrix}\begin{bmatrix}\Sigma_{11}s_{\zeta 1\zeta 1} & \Sigma_{12}s_{\zeta 1\zeta 2} & \dots \\ \Sigma_{21}s_{\zeta 2\zeta 1} & \Sigma_{22}s_{\zeta 2\zeta 2} & \dots \\ \vdots & \ddots & \\ \dots & \dots & \Sigma_{GG}s_{\zeta G\zeta G}\end{bmatrix}^{-1}\begin{bmatrix}s'_{\xi 1\zeta 1} & 0 & \dots \\ 0 & s'_{\xi 2\zeta 2} & \dots \\ \vdots & \ddots & \\ \dots & \dots & s'_{\xi G\zeta G}\end{bmatrix}\right]^{-1}}_{:=A_{\text{3SLS}}^{-1}} \\ \underbrace{\begin{bmatrix}s_{\xi 1\zeta 1} & 0 & \dots \\ 0 & s_{\xi 2\zeta 2} & \dots \\ \vdots & \ddots & \\ \dots & \dots & s_{\xi G\zeta G}\end{bmatrix}\begin{bmatrix}\Sigma_{11}s_{\zeta 1\zeta 1} & \Sigma_{12}s_{\zeta 1\zeta 2} & \dots \\ \Sigma_{21}s_{\zeta 2\zeta 1} & \Sigma_{22}s_{\zeta 2\zeta 2} & \dots \\ \vdots & \ddots & \\ \dots & \dots & \Sigma_{GG}s_{\zeta G\zeta G}\end{bmatrix}^{-1}\begin{bmatrix}\psi_1 & 0 & \dots \\ 0 & \psi_2 & \dots \\ \vdots & \ddots & \\ \dots & \dots & \psi_G\end{bmatrix}}_{:=B_{\text{3SLS}}}\Delta,\end{aligned} \tag{68}$$

where $s_{\xi i\zeta i} = \text{plim}_{N\to\infty}\frac{\hat{\xi}_i'\hat{\zeta}_i}{N}$, $s_{\zeta i\zeta j} = \text{plim}_{N\to\infty}\frac{\hat{\zeta}_i'\hat{\zeta}_j}{N}$, and $\psi_i = \text{plim}_{N\to\infty}\frac{\vec{\hat{\zeta}}'\eta_{\xi i}}{N}$, and i and j are index variables. The more complex expressions for A_{3SLS} and B_{3SLS} in Equation (68) simplify to the expressions given in Equation (61) for the case of $\zeta \equiv \zeta_1 = \zeta_2 = \cdots = \zeta_G$.

Updating our definitions of $\hat{A}_{\text{3SLS}}$, $\hat{B}_{\text{3SLS}}$, and $\widehat{\text{BiasFactor}}_{\text{3SLS}}$ to match the general case, our proposed consistent estimator, $D_{\text{3SLS}}^{\text{con}}$, is obtained by solving the linear system of equations:

$$\left[I_{Gg} - \widehat{\text{BiasFactor}}_{\text{3SLS}}\right] D_{\text{3SLS}}^{\text{con}} = D_{\text{3SLS}}. \tag{69}$$

Or, equivalently:

$$D_{3\text{SLS}}^{\text{con}} = \left[I_{Gg} - \widehat{\text{BiasFactor}}_{3\text{SLS}} \right]^{-1} D_{3\text{SLS}}. \tag{70}$$

Again, by Slutsky's theorem and the continuous mapping theorem, it follows that $D_{3\text{SLS}}^{\text{con}}$ converges in probability to Δ in the general case:

$$\operatorname*{plim}_{N\to\infty} D_{3\text{SLS}}^{\text{con}} = \Delta. \tag{71}$$

D.4 Asymptotic Normality and Inference

Having established that our bias-corrected estimator is consistent, we now demonstrate that it is also asymptotically normal and, hence, suitable for standard inference procedures. We can write the probability limit of the naïve estimator as

$$\operatorname*{plim}_{N\to\infty} D_{3\text{SLS}} = \left(I_{Gg} - A_{3\text{SLS}}^{-1} B_{3\text{SLS}} \right) \Delta. \tag{72}$$

Moreover, the standard (naïve) 3SLS estimator satisfies

$$\sqrt{N} \left[D_{3\text{SLS}} - \left(I_{Gg} - A_{3\text{SLS}}^{-1} B_{3\text{SLS}} \right) \Delta \right] \xrightarrow{d} \mathcal{N}(0,\, V_{3\text{SLS}}), \tag{73}$$

where $V_{3\text{SLS}}$ denotes the asymptotic variance of the naïve 3SLS estimator.

Since the correction function

$$h_{3\text{SLS}}(D_{3\text{SLS}}) = \left(I_{Gg} - \widehat{\text{BiasFactor}}_{3\text{SLS}} \right)^{-1} D_{3\text{SLS}} \tag{74}$$

is continuously differentiable in a neighborhood of the probability limit $\left(I_{Gg} - A_{3\text{SLS}}^{-1} B_{3\text{SLS}} \right) \Delta$, and because $\widehat{\text{BiasFactor}}_{3\text{SLS}}$ converges in probability to $A_{3\text{SLS}}^{-1} B_{3\text{SLS}}$, the continuous mapping theorem

and the delta method imply that

$$\sqrt{N}\left(D_{3\text{SLS}}^{\text{con}} - \Delta\right) = \sqrt{N}\left[\left(I_{Gg} - \widehat{\text{BiasFactor}}_{3\text{SLS}}\right)^{-1} D_{3\text{SLS}} \; - \; \Delta\right]$$
$$\xrightarrow{d} \; \mathcal{N}\Big(0, \; \left(I_{Gg} - A_{3\text{SLS}}^{-1} B_{3\text{SLS}}\right)^{-1} V_{3\text{SLS}} \left[\left(I_{Gg} - A_{3\text{SLS}}^{-1} B_{3\text{SLS}}\right)^{-1}\right]'\Big). \tag{75}$$

In other words, although the naïve estimator $D_{3\text{SLS}}$ is biased due to AI-coding errors, our correction—by premultiplying with $(I_{Gg} - \widehat{\text{BiasFactor}}_{3\text{SLS}})^{-1}$—removes that bias and, via a first-order Taylor expansion of $h_{3\text{SLS}}(D_{3\text{SLS}})$, preserves asymptotic normality with a transformed asymptotic covariance matrix. This result justifies the use of conventional statistical inference tools—such as hypothesis testing and the construction of confidence intervals—either by using the delta method or, as we recommend in practice, via bootstrap methods that consistently approximate the distribution of $D_{3\text{SLS}}^{\text{con}}$.

D.5 Algorithm

$A_{3\text{SLS}}$ and $B_{3\text{SLS}}$ are defined in Equations (61) and (68) for $\zeta \equiv \zeta_1 = \zeta_2 = \cdots = \zeta_G$ and the general case respectively. Data_s denotes the manually coded subsample, and Data_f denotes the full sample. N_s and N_f represent the number of draws from Data_s and Data_f, respectively.

Algorithm 7 Algorithm for Inference in System-of-Equations Marketing Models

Input: Manually coded calibration subsample (Data_s), AI-coded full sample (Data_f).
▷ Steps 1-3 (data acquisition, manual coding, AI coding) are assumed to be complete.
for $b \leftarrow 1, nB$ **do**
Draw bootstrap sample Bootstrap Subsample$[b]$ of size N_s from Data_s.
Infer $\hat{\psi}_g = \frac{\vec{\hat{\zeta}}' \eta_{\xi g}}{N_s}, g = 1, \ldots G$ in Bootstrap Subsample$[b]$.
Draw bootstrap sample Bootstrap Full Sample$[b]$ of size N_f from Data_f.
Compute the 3SLS estimator $D_{3\text{SLS}}[b]$ in Bootstrap Full Sample$[b]$.
Infer $\hat{s}_{\xi_i \zeta}$ for $i = 1, \ldots, G$ and $\hat{s}_{\zeta\zeta}$ in Bootstrap Full Sample$[b]$.
Infer the estimation analogs of $A_{3\text{SLS}}$ and $B_{3\text{SLS}}$: $\hat{A}_{3\text{SLS}}$ and $\hat{B}_{3\text{SLS}}$.
Compute $\widehat{\text{BiasFactor}}_{3\text{SLS}} = \hat{A}_{3\text{SLS}}^{-1} \hat{B}_{3\text{SLS}}$.
Solve $\left[I_{Gg} - \widehat{\text{BiasFactor}}_{3\text{SLS}}\right] D_{3\text{SLS}}^{\text{con}}[b] = D_{3\text{SLS}}[b]$ in Bootstrap Full Sample$[b]$.
end for
Infer test statistics from the distribution of $D_{3\text{SLS}}^{\text{con}}[b], \quad b = 1, \ldots, nB$.

E Addressing Imprecision in the Manually-coded Measures

The preceding derivations assume that manual coding in the calibration subsample provides a "ground truth" free of error. In practice, however, even careful manual coding introduces some imprecision. Human coders—even when well-trained—inevitably exhibit variability in their ratings, particularly for subjective constructs. For instance, if 20 coders rate an advertisement with a true nostalgia level of 5 (on a 1–7 scale) and each coder's rating has a standard deviation of 1 point, the average of their ratings will still have a standard deviation of approximately 0.22. Thus, while the manually coded "ground truth" is an unbiased estimate of the true value, it remains imprecise, following a normal distribution centered on the true value.

This result aligns with established assumptions in the econometrics literature on measurement error, where error in human-coded measures is traditionally treated as additive Gaussian noise (or a transformation thereof) (e.g., Abel 2017). In these approaches, measurement error in human-coded variables is assumed to be orthogonal to the regressors, since it typically arises from idiosyncratic, coder-specific factors rather than being systematically linked to the research question of interest.

We begin by observing that our estimators are relatively robust to such imprecision in manual coding. This robustness arises because, in Equations (37) and (72), the estimated cross-covariance matrix of the observed instruments and the coding errors in the regressors (denoted ψ_{2SLS} in 2SLS and ψ_g, $g = 1, \ldots, G$, in 3SLS) incorporates manual coding error only linearly. Consequently, the inferred cross-covariance matrices remain stable even when manual coding is imprecise. Intuitively, noise from processes such as sampling and inter-coder variability tends to average out, reducing its impact on inference.

However, in theory, this imprecision introduces an additional source of potential bias. The estimators we proposed in prior subsections address AI-coding error, but they rely on a consistent estimate of ψ_{2SLS} from the manually coded subsample (or ψ_g in the 3SLS case). If the manually coded variables in this subsample contain imprecision, then $\hat{\psi}_{\text{2SLS}}$ (or $\hat{\psi}_g$) may itself be estimated imprecisely, potentially introducing residual bias into the final estimates.

E.1 Mathematical Formulation

To address this issue more comprehensively, we adapt the Simulation-Extrapolation (SIMEX) method of Cook and Stefanski (1994). The central idea of SIMEX is to simulate the effect of coding error by artificially adding noise, estimating the model at different noise levels, and then extrapolating back to the case of no error. This process, in the context of our estimators, can be represented mathematically as follows.

Consider a sequence of datasets indexed by λ_i, where $i = 0, 1, \ldots$ with $0 = \lambda_0 < \lambda_i$ for $i \geq 1$. For each dataset, we define:

$$\xi_s(\lambda) = \xi_s + \lambda\, \mathcal{N}(0, 1), \tag{76}$$

$$\zeta_s(\lambda) = \zeta_s + \lambda\, \mathcal{N}(0, 1), \tag{77}$$

where ξ_s and ζ_s are the manually coded values in subsample s. That is, for each element of ξ_s, we add an independent draw from $\mathcal{N}(0, 1)$ scaled by λ, and similarly for ζ_s.

We denote $i^* \geq 0$ as the index value at which the imprecision in the observed manually coded measures is distributionally equivalent (i.e., having the same statistical properties, such as mean, variance, and distributional shape) to that in the dataset indexed by i^*. In other words, λ_{i^*} is the level of added noise that matches the imprecision inherent in the observed manually coded data. This can be determined, for instance, by estimating the standard deviation of coding errors in s and setting λ_{i^*} equal to this estimated value, thereby ensuring that the artificially generated noise matches the observed variability in manual coding.

Datasets with greater imprecision—namely, $\xi_s(\lambda_j)$ and $\zeta_s(\lambda_j)$ for $j = i^*, i^* + 1, \ldots$—can be

constructed by adding further noise:

$$\xi_s(\lambda_i) = \xi_s(\lambda_{i^*}) + (\lambda_i - \lambda_{i^*})\ \mathcal{N}(0,1), \quad i \geq i^*, \tag{78}$$

$$= \hat{\xi}_s + (\lambda_i - \lambda_{i^*})\ \mathcal{N}(0,1), \quad i \geq i^*, \tag{79}$$

$$\zeta_s(\lambda_i) = \zeta_s(\lambda_{i^*}) + (\lambda_i - \lambda_{i^*})\ \mathcal{N}(0,1), \quad i \geq i^*, \tag{80}$$

$$= \hat{\zeta}_s + (\lambda_i - \lambda_{i^*})\ \mathcal{N}(0,1), \quad i \geq i^*. \tag{81}$$

Datasets with less imprecision (i.e., for $j = 0, \ldots, i^* - 1$) cannot be constructed from the observed data; instead, estimates corresponding to such levels must be extrapolated. To that end, let $\hat{\theta}(\lambda)$ denote an estimator based on $\xi_s(\lambda)$ and $\zeta_s(\lambda)$, such that

$$\lim_{\lambda \to 0} \hat{\theta}(\lambda) = \hat{\theta}(0) = \theta, \tag{82}$$

where θ represents the true parameters.

The estimators we propose (for both single-equation models and systems of equations) satisfy this condition[13]. As coding error in our manually coded variables tends to zero (i.e., $\lim_{\lambda \to 0} \xi_s(\lambda) \to \xi_s$ and $\lim_{\lambda \to 0} \zeta_s(\lambda) \to \zeta_s$), the bias introduced by manual coding error vanishes asymptotically.

E.2 Doubly Robust for Addressing Imprecision in the Manually Coded Measures

Based on this result, we propose a novel two-step method. First, in a simulation step, we construct $\xi_s(\lambda_i)$ and $\zeta_s(\lambda_i)$ for increasing values of i, and therefore for increasing values of λ_i. For each i, we estimate $\hat{\delta}(\lambda_i)$. Second, in a back-extrapolation step, we infer $\hat{\delta}(\lambda_0)$ from $\hat{\delta}(\lambda_i)$. We depict this method in Algorithm 8, where nB represents the number of bootstrap iterations, Data_s denotes the manually coded subsample, and Data_f refers to the full sample.

[13] This result follows from a generalization of the continuous mapping theorem to M-estimators by Seijo and Sen (2011).

Algorithm 8 Doubly Robust Algorithm: Addressing Imprecision in the Manually Coded Measures

Input: Manually coded calibration subsample (Data_s), AI-coded full sample (Data_f).
▷ Steps 1-3 (data acquisition, manual coding, AI coding) are assumed to be complete.
for $k \leftarrow 1, K$ **do**
  for $b \leftarrow 1, nB$ **do**
    Draw bootstrap sample Bootstrap Subsample$[b]$ of size N_s from Data_s.
    Simulate imprecision corresponding to k and add to the manually coded measures in Bootstrap Subsample$[b]$.
    Infer limit matrices and densities in Bootstrap Subsample$[b]$.
    Draw bootstrap sample Bootstrap Full Sample$[b]$ of size N_f from Data_f.
    Compute $d_{2\text{SLS}}[b,k]$ or $D_{3\text{SLS}}[b,k]$ using limit matrices in Bootstrap Full Sample$[b]$.
  end for
  Compute critical values of $d_{2\text{SLS}}[,k]$ or $D_{3\text{SLS}}[,k]$ as d_k or D_k.
end for
Model $d_k, k = 1, \ldots K$ as a function of k and extrapolate to the case of no imprecision.

E.3 Distinction from Prior Estimators and Contributions

Algorithm 3 extends the original formulation and subsequent adaptations of SIMEX (e.g., Cook and Stefanski 1994; Yang et al. 2018) to simultaneously account for *both* AI-coding error and manual-coding error. Prior SIMEX estimators were designed to correct for a single type of measurement error, meaning they can be applied to either AI-coding errors or manual-coding errors, but not both at the same time. In contrast, our method is explicitly designed to address both sources of error within a unified framework.

A key limitation of SIMEX models is their reliance on the assumption that measurement error follows an additive, scaled parametric distribution—e.g., a Gaussian distribution in Cook and Stefanski (1994) or a linear transformation of a Gaussian distribution in Yang et al. (2018). These assumptions are well-suited for manually coded measures, where independent ratings from multiple human coders can be averaged, and the central limit theorem ensures that such averages approximate a Gaussian distribution. However, as previously discussed, AI-coded variables often deviate from true values in more complex, systematic ways that do not conform to these assumptions. In particular, AI-coding errors may be correlated with other variables, exhibit heteroskedasticity, or depend on the specific characteristics of the AI model used for coding.

As a result, SIMEX models are better suited for human-coded measurement error—such as that encountered in surveys—than for AI-coding errors. However, they remain a natural candidate for addressing human-coding error in the calibration subsample, where manually coded data are used to train and validate AI models. Our method explicitly incorporates this distinction by imposing fewer restrictions on AI-coding errors. Specifically, we only require that the first and second moments of the AI-coding error exist and be well-defined, rather than assuming a specific parametric distribution. Instead, we apply SIMEX assumptions to address imprecision in the manually coded measures. This distinction enhances the conceptual compatibility of our method with AI coding in large-scale marketing datasets, where AI-generated variables may exhibit non-classical error structures.

Despite these advances, the accuracy of our estimates depends critically on the precision of the manually coded measures. If the manually coded data contain substantial imprecision, our method must back-extrapolate across a wide parameter range, potentially reducing accuracy. Conversely, when the manually coded measures are highly precise and closely approximate the true values, less extrapolation is required, leading to nearly consistent estimates. This relationship between manual-coding precision and estimator accuracy underscores the importance of obtaining high-quality manually coded data for the calibration subsample, even when working with AI-coded variables at scale. We illustrate this trade-off empirically in our simulations, providing concrete evidence of how varying levels of imprecision in manually coded measures affect inference.

F Monte Carlo Simulations: R Code Implementations

Below are the R code implementations of the simulation models presented in the main manuscript, designed to validate the proposed bias-corrected estimators for AI-coded variables in both single-equation and system-of-equations marketing models. The simulations create synthetic datasets that mimic real-world scenarios where researchers use AI to code key marketing constructs (e.g., nostalgia, product features, brand heritage) from unstructured data. These constructs are then incorporated into econometric models to estimate their impact on consumer behavior outcomes (e.g., engagement, brand attitude). The code is structured to allow for easy modification of parameters, enabling researchers to explore the performance of the estimators under various conditions, including different levels of AI-coding error, endogeneity, and imprecision in manually coded measures. The simulations are divided into three main parts: single-equation model simulations, system-of-equations model simulations, and robustness/extrapolation simulations.

The single-equation model simulation code implements four distinct study scenarios. Study 1 serves as a baseline, with all regressors exogenous and measured without error. Study 2 introduces AI-coding error in the focal independent variable (nostalgia intensity) that is correlated with an observed covariate (a product feature). Study 3 adds endogeneity to the focal variable, requiring the use of instrumental variables (brand heritage and cost of producing nostalgic content). Study 4 further complicates the scenario by introducing AI-coding error in the instruments themselves. For each study, the code generates data from a multivariate normal distribution, with the covariance matrix carefully specified to induce the desired correlations and error structures. It then estimates the model parameters using ordinary least squares (OLS), two-stage least squares (2SLS), and the proposed bias-corrected 2SLS estimator. A bootstrap procedure is used to obtain standard errors for the corrected estimator.

The system-of-equations model simulation code extends the single-equation model to a two-equation model, representing scenarios where multiple related outcomes (engagement and brand attitude) are jointly influenced by common factors. The code implements four study scenarios

analogous to the single-equation case, with the added complexity of correlated errors across equations. The data generation process is similar, using a multivariate normal distribution, but with a covariance structure that induces correlations between the error terms of the two equations, as well as potential endogeneity and AI-coding errors. The code estimates the model parameters using three-stage least squares (3SLS) and the proposed bias-corrected 3SLS estimator, again employing a bootstrap procedure for standard error estimation.

The robustness and extrapolation simulation code addresses the potential for imprecision in the manually coded calibration subsample, which is used to estimate the key parameters for the bias correction. It implements the Simulation-Extrapolation (SIMEX) method to correct for biases introduced by manual coding errors. The code systematically adds increasing levels of synthetic noise to the manually coded variables, re-estimates the model parameters (including the bias-corrected estimators) for each noise level, and then extrapolates back to the case of no added noise using a Generalized Additive Model (GAM). This entire process is embedded within a bootstrap loop to obtain standard errors for the SIMEX-corrected estimates. The code produces plots that visualize the impact of manual coding imprecision on the estimates and demonstrate the effectiveness of the SIMEX correction.

F.1 Single-Equation Model Simulation

Below is a description of how the key constructs in the manuscript and web appendix map to the variables in the single-equation simulation R code. Table A8 provides a concise summary.

Notation	Code	Description
y	`y`	Dependent variable (e.g., engagement).
z_{nost}	`xi[, 1]` (true), `x[, 1]` (observed)	Focal AI-coded regressor (e.g., nostalgia).
x_{fea}	`xi[, 2]` (true), `x[, 2]` (observed)	Observed covariate (e.g., product feature).
w_{her}, w_{cost}	`z[, 2]`, `z[, 3]`	Instrumental variables (e.g., heritage, cost).
ε	`epsilon`	Structural error term.
η_z	`draws[, 10]`	AI-coding error in z_{nost}.
η_w	`draws[, 12]`, `draws[, 13]`	AI-coding error in instruments (Study 4).
ξ	`xi`	Matrix of true regressors.
$\hat{\xi}$	`x`	Matrix of observed regressors.
ζ	`z`	Matrix of true instruments.
$\hat{\zeta}$	`z`	Matrix of observed instruments.
ψ	`bs_Omega`	Cross-product matrix of instruments and coding errors.
d	`b_2sls_big`	Canonical 2SLS estimator.
d^{con}	`bootstrap_b_2sls_corr[ind, ]`	Bias-corrected 2SLS estimator.
$\widehat{\text{BiasFactor}}$	`correction_matrix`	Estimated bias factor.

Table A8: Mapping the Mathematical Notation (Manuscript and Web Appendix) to R Variable Names

The dependent variable, denoted as y in the manuscript (e.g., consumer engagement), is represented by the variable `y` in the code. The underlying random components for all variables (including exogenous components, endogenous components, and error terms) are initially drawn from a multivariate normal distribution and stored in the `draws` matrix. The true, unobserved values of the regressors are stored in the matrix `xi`. Specifically, `xi[, 1]` corresponds to the true value of the focal AI-coded regressor, z_{nost} (nostalgia intensity), *before* any endogeneity or measurement error is added. `xi[, 2]` corresponds to the true value of the observed covariate, x_{fea} (product feature), which is assumed to be measured without error. The *observed* regressors, which may include AI-coding error, are stored in the matrix `x`. In scenarios with measurement error in z_{nost}, `x[, 1]` represents the observed, noisy version of nostalgia intensity, while `x[, 2]` remains identical to `xi[, 2]`. The instruments are represented by the variable `z`. In studies

1 and 2, where there is no endogeneity or instrument error, `z` is simply equal to `x`. In studies 3 and 4, `z` contains the instruments: `z[, 1]` corresponds to the exogenous component of x_{fea} or x_{fea} itself, `z[, 2]` corresponds to w_{her} (brand heritage), and `z[, 3]` corresponds to w_{cost} (cost of producing nostalgic content). The structural error term, ε, is represented by the variable `epsilon`. The AI-coding errors themselves are generated as part of the `draws` matrix, with column 10 (`draws[, 10]`) representing the error for z_{nost}, and columns 12 and 13 (`draws[, 12]` and `draws[, 13]`) representing the errors for w_{her} and w_{cost}, respectively (in Study 4). The key matrix for the bias correction, ψ (the cross-product of instruments and coding errors), is estimated within the bootstrap loop and stored in the variable `bs_Omega`, *which is calculated within each bootstrap iteration using the calibration subsample.* The canonical 2SLS estimator is represented by `b_2sls_big`, and the bias-corrected 2SLS estimator is stored in `bootstrap_b_2sls_corr[ind, ]`.

```

# Load the required library for multivariate normal distributions
library(mvtnorm)

# Clear the workspace except for 'sim_ind' and 'data_imprecision'.
# 'sim_ind' is used to select a specific simulation run when doing robustness checks.
# 'data_imprecision' holds the parameters for robustness simulations.
rm(list = setdiff(ls(), c("sim_ind", "data_imprecision")))

# Ensure 'sim_ind' exists; if not, set it to 1 (default for standard simulations).
if (!exists("sim_ind")) {
  sim_ind <- 1
}

# Check if 'data_imprecision' exists (this indicates a robustness simulation run).
if (!exists("data_imprecision")) {
  # If 'data_imprecision' doesn't exist, set default simulation parameters for the
  # standard simulation (no added imprecision in manual coding).
  imprecision <- 0             # No added imprecision
  study_range <- 1:4           # Run all four study scenarios
```

```r
  n_total <- 50000              # Total number of observations
  n_subsample <- 2500           # Size of the calibration subsample
   ↪  (with manual coding)
} else {
  # If 'data_imprecision' exists, use its values for a robustness
   ↪  simulation run.
  imprecision <- data_imprecision[sim_ind, "imprecision"] # Get
   ↪  imprecision level
  study_range <- 4              # Only run Study 4 (most complex case)
   ↪  for robustness analysis
  n_total <- 50000              # Total number of observations
  n_subsample <- 2500           # Size of the calibration subsample
}

# Function to simulate data based on the specified study scenario
simulate_data <- function(n_total, study) {
  set.seed(102024)              # Set seed for reproducibility

  # Initialize mean vector and covariance matrix for multivariate
   ↪  normal distribution.
  # All variables are initially centered at 0.
  mu <- rep(0, 13)              # Mean vector (zeros for all variables)
  sigma <- diag(13)             # Covariance matrix (identity matrix by
   ↪  default)

  # Variable indices in 'draws' (the matrix of simulated data):
  # [1]  epsilon       : Structural error term (for the dependent
   ↪  variable).
  # [2]  z_nost_exog   : Exogenous component of z_nost (nostalgia).
  # [3]  x_fea_exog    : Exogenous component of x_fea (product
   ↪  feature).
  # [4]  w_her_exog    : Exogenous component of w_her (brand heritage
   ↪  instrument).
  # [5]  w_cost_exog   : Exogenous component of w_cost (cost of
   ↪  nostalgia instrument).
  # [6]  z_nost_endog  : Endogenous component of z_nost (correlated
   ↪  with epsilon).
  # [7]  (Unused)      : Placeholder (not used in the current
   ↪  single-equation model).
  # [8]  (Unused)      : Placeholder.
  # [9]  (Unused)      : Placeholder.
  # [10] z_nost_me     : Measurement error in z_nost (AI coding error).
  # [11] (Unused)      : Placeholder.
  # [12] w_her_me      : Measurement error in w_her (AI coding error).
```

```r
  # [13] w_cost_me     : Measurement error in w_cost (AI coding error).

  # Adjust the covariance matrix 'sigma' based on the study number to
  ↪ introduce
  # specific correlations and error structures.
  if (study == 1) {
    # Study 1: All regressors are exogenous and measured without error.
    # This is the baseline scenario.
    draws <- rmvnorm(n_total, mean = mu, sigma = sigma)  # Generate
    ↪ data

    # True regressors: z_nost (nostalgia) and x_fea (feature).
    xi <- draws[, c(2, 3)]       # Select columns for z_nost_exog and
    ↪ x_fea_exog

    # Observed regressors are the same as true regressors (no
    ↪ measurement error).
    x <- xi

    # Instruments (zeta) are the same as observed regressors in this
    ↪ case.
    z <- x

  } else if (study == 2) {
    # Study 2: Measurement error in z_nost (nostalgia) correlated with
    ↪ x_fea (feature).
    sigma[3, 10] <- sigma[10, 3] <- 0.4  # Correlation between
    ↪ x_fea_exog and z_nost_me
    draws <- rmvnorm(n_total, mean = mu, sigma = sigma)

    # True regressors: z_nost and x_fea.
    xi <- draws[, c(2, 3)]

    # Observed regressors (x) with measurement error in z_nost.
    x <- xi
    x[, 1] <- x[, 1] + draws[, 10]       # Add measurement error to
    ↪ z_nost

    # Instruments (zeta) are the same as observed regressors.
    z <- x

  } else if (study == 3) {
    # Study 3: Endogeneity in z_nost (nostalgia); instruments without
    ↪ measurement error.
```

```r
    sigma[3, 10] <- sigma[10, 3] <- 0.4  # Correlation between x_fea_exog and z_nost_me
    sigma[1, 6] <- sigma[6, 1] <- 0.2    # Correlation between epsilon and z_nost_endog
    sigma[2, 4] <- sigma[4, 2] <- 0.3    # Correlation between z_nost_exog and w_her_exog
    sigma[2, 5] <- sigma[5, 2] <- 0.7    # Correlation between z_nost_exog and w_cost_exog

    draws <- rmvnorm(n_total, mean = mu, sigma = sigma)

    # True regressors: z_nost and x_fea, with endogenous component added to z_nost.
    xi <- draws[, c(2, 3)]
    xi[, 1] <- xi[, 1] + draws[, 6]      # Add endogenous component to z_nost

    # Observed regressors (x) with measurement error in z_nost.
    x <- xi
    x[, 1] <- x[, 1] + draws[, 10]       # Add measurement error to z_nost

    # Instruments (zeta) are x_fea_exog, w_her_exog, w_cost_exog (without measurement error).
    z <- draws[, c(3, 4, 5)]

  } else if (study == 4) {
    # Study 4: Endogeneity in z_nost (nostalgia); measurement error in instruments
    # correlated with x_fea (feature).
    sigma[3, 10] <- sigma[10, 3] <- 0.4  # Correlation between x_fea_exog and z_nost_me
    sigma[1, 6] <- sigma[6, 1] <- 0.2    # Correlation between epsilon and z_nost_endog
    sigma[2, 4] <- sigma[4, 2] <- 0.3    # Correlation between z_nost_exog and w_her_exog
    sigma[2, 5] <- sigma[5, 2] <- 0.7    # Correlation between z_nost_exog and w_cost_exog
    sigma[3, 12] <- sigma[12, 3] <- 0.3  # Correlation between x_fea_exog and w_her_me
    sigma[3, 13] <- sigma[13, 3] <- 0.3  # Correlation between x_fea_exog and w_cost_me

    draws <- rmvnorm(n_total, mean = mu, sigma = sigma)
```

```r

    # True regressors: z_nost and x_fea, with endogenous component
    ↪ added to z_nost.
    xi <- draws[, c(2, 3)]
    xi[, 1] <- xi[, 1] + draws[, 6]  # Add endogenous component to
    ↪ z_nost

    # Observed regressors (x) with measurement error in z_nost.
    x <- xi
    x[, 1] <- x[, 1] + draws[, 10]  # Add measurement error to z_nost

    # Instruments (zeta) are x_fea_exog, w_her_exog, w_cost_exog with
    ↪ measurement error.
    z <- draws[, c(3, 4, 5)]
    z[, 2] <- z[, 2] + draws[, 12]        # Add measurement error to
    ↪ w_her_exog (heritage)
    z[, 3] <- z[, 3] + draws[, 13]        # Add measurement error to
    ↪ w_cost_exog (cost)

  } else {
    stop("Study setup not defined") # Error if an invalid study number
    ↪ is provided.
  }

  # Structural error term (epsilon).
  epsilon <- draws[, 1]

  # Dependent variable: y = z_nost * beta_nost + x_fea * gamma_fea +
  ↪ epsilon.
  # The coefficients beta_nost and gamma_fea are implicitly set to 1.
  y <- rowSums(xi) + epsilon

  # Generate coding imprecision deviates (standard normal distribution)
  ↪ for the
  # *manually coded* subsample.  This is used for the SIMEX procedure
  ↪ (robustness analysis).
  imprecision_vec <- rnorm(n_subsample)

  # Return a list containing all generated variables.
  return(list(
    epsilon = epsilon,  # Structural error
    xi = xi,            # True regressors
    x = x,              # Observed regressors (with AI coding error)
    z = z,              # Instruments (with AI coding error in Study 4)
```

```r
    y = y,                # Dependent variable
    imprecision_vec = imprecision_vec # Vector of imprecision deviates
     ↪  for SIMEX
  ))
}

# Initialize matrix to store coefficients and standard errors
# Rows: estimates and standard errors for each estimator (OLS, 2SLS,
 ↪  Corrected 2SLS)
# Columns: studies 1 to 4
coef_mat <- matrix(0, 12, 4)

# Loop over the specified studies
for (study in study_range) {
  # Simulate data for the current study
  sim_data <- simulate_data(n_total, study)
  epsilon <- sim_data$epsilon
  xi <- sim_data$xi
  x <- sim_data$x
  z <- sim_data$z  # Now consistently 'z', even when it equals 'x'
  y <- sim_data$y
  imprecision_vec <- sim_data$imprecision_vec

  # Ordinary Least Squares (OLS) estimation
  b_ols <- solve(t(x) %*% x) %*% t(x) %*% y                              #
   ↪  OLS coefficients
  residuals_ols <- y - x %*% b_ols                                        #
   ↪  Residuals
  b_ols_sd <- sqrt(diag(solve(t(x) %*% x) * sum(residuals_ols^2) /
   ↪  (n_total - ncol(x))))  # Standard errors

  # Two-Stage Least Squares (2SLS) estimation
  b_2sls <- solve(t(x) %*% z %*% solve(t(z) %*% z) %*% t(z) %*% x) %*%
    t(x) %*% z %*% solve(t(z) %*% z) %*% t(z) %*% y                      #
     ↪  2SLS coefficients
  residuals_2sls <- y - x %*% b_2sls                                      #
   ↪  Residuals
  b_2sls_sd <- sqrt(diag(solve(t(x) %*% z %*% solve(t(z) %*% z) %*%
   ↪  t(z) %*% x) *
                         sum(residuals_2sls^2) / (n_total -
                          ↪  ncol(x))))    # Standard errors

  # Store OLS estimates and standard errors
  coef_mat[1, study] <- b_ols[1]           # beta_nost estimate
```

```r
  coef_mat[2, study] <- b_ols_sd[1]        # beta_nost standard error
  coef_mat[7, study] <- b_ols[2]           # gamma_fea estimate
  coef_mat[8, study] <- b_ols_sd[2]        # gamma_fea standard error

  # Store 2SLS estimates and standard errors
  coef_mat[3, study] <- b_2sls[1]
  coef_mat[4, study] <- b_2sls_sd[1]
  coef_mat[9, study] <- b_2sls[2]
  coef_mat[10, study] <- b_2sls_sd[2]

  # Bootstrap procedure for corrected 2SLS estimates
  bootstrap_b_2sls_corr <- matrix(0, 1000, 2)  # Initialize matrix
  ↪  (1000 bootstrap samples, 2 coefficients)

  for (ind in 1:1000) {
    # Bootstrap sample for the calibration subsample (with replacement)
    subsample_indices <- sample(1:n_subsample, n_subsample, replace =
    ↪  TRUE)
    xi_imprecise_subsample <- xi[subsample_indices, ]
    # Add coding imprecision to z_nost in the subsample (for SIMEX, if
    ↪  imprecision > 0)
    xi_imprecise_subsample[, 1] <- xi_imprecise_subsample[, 1] +
    ↪  imprecision_vec * imprecision

    # Calculate bs_Omega matrix (differences between observed and true
    ↪  regressors).
    # This is the key matrix for the bias correction, estimated using
    ↪  the
    # *calibration subsample* where we have both the "true" (manually
    ↪  coded) values
    # and the AI-coded (or observed, with error) values.  This
    ↪  corresponds to
    # \hat{\psi} in the manuscript and web appendix.
    bs_Omega <- t(z[subsample_indices, ]) %*% (x[subsample_indices, ]
    ↪  - xi_imprecise_subsample) / n_subsample

    # Bootstrap sample for the larger dataset (the remaining data,
    ↪  *not* the calibration subsample)
    total_indices <- sample((n_subsample + 1):n_total, n_total -
    ↪  n_subsample, replace = TRUE)
    big_x <- x[total_indices, ]  # Observed regressors (with AI coding
    ↪  error)
    big_z <- z[total_indices, ]  # Observed instruments (with AI coding
    ↪  error in Study 4)
```

```r
    big_y <- y[total_indices]    # Dependent variable

    # 2SLS estimation on the larger dataset (using the AI-coded/observed data).
    # This corresponds to 'd' in Equation 14 of the Web Appendix.
    b_2sls_big <- solve(t(big_x) %*% big_z %*% solve(t(big_z) %*% big_z) %*% t(big_z) %*% big_x) %*%
      t(big_x) %*% big_z %*% solve(t(big_z) %*% big_z) %*% t(big_z) %*% big_y

    # Correction matrix for the estimator. This uses bs_Omega (estimated from the
    # calibration subsample) to correct for the bias in the 2SLS estimator.
    # This corresponds to (I_g - \widehat{BiasFactor}) in Equation 17 of the Web Appendix.
    correction_matrix <- diag(ncol(x)) - (n_total - n_subsample) * solve(
      t(big_x) %*% big_z %*% solve(t(big_z) %*% big_z) %*% t(big_z) %*% big_x
    ) %*% t(big_x) %*% big_z %*% solve(t(big_z) %*% big_z) %*% bs_Omega

    # Corrected 2SLS estimates. This corresponds to d^{con} in Equation 17.
    bootstrap_b_2sls_corr[ind, ] <- solve(correction_matrix) %*% b_2sls_big
  }

  # Store corrected 2SLS estimates and standard errors (median and SD from bootstrap samples)
  coef_mat[5, study] <- median(bootstrap_b_2sls_corr[, 1]) # Median of beta_nost estimates
  coef_mat[6, study] <- sd(bootstrap_b_2sls_corr[, 1])     # SD of beta_nost estimates
  coef_mat[11, study] <- median(bootstrap_b_2sls_corr[, 2]) # Median of gamma_fea estimates
  coef_mat[12, study] <- sd(bootstrap_b_2sls_corr[, 2])     # SD of gamma_fea estimates
}

# Output the results in LaTeX table format if 'data_imprecision' doesn't exist
# (i.e., if we're not doing the robustness analysis with SIMEX).
if (!exists("data_imprecision")) {
```

```
  cat(
    paste(
      # Coefficients for beta_nost (nostalgia)
      paste(
        "\\multirow{6}{*}{$\\beta_{\\text{nost}}$} &
         ↪  \\multirow{2}{*}{Regression} &",
        paste(round(coef_mat[1, ], 3), collapse = " & "),
        "\\\\"
      ),
      paste(
        "& &",
        paste(paste("(", round(coef_mat[2, ], 3), ")", sep = ""),
         ↪  collapse = " & "),
        "\\\\"
      ),
      paste(
        "& \\multirow{2}{*}{2SLS} &",
        paste(round(coef_mat[3, ], 3), collapse = " & "),
        "\\\\"
      ),
      paste(
        "& &",
        paste(paste("(", round(coef_mat[4, ], 3), ")", sep = ""),
         ↪  collapse = " & "),
        "\\\\"
      ),
      paste(
        "& \\multirow{2}{*}{2SLS Corrected} &",
        paste(round(coef_mat[5, ], 3), collapse = " & "),
        "\\\\"
      ),
      paste(
        "& &",
        paste(paste("(", round(coef_mat[6, ], 3), ")", sep = ""),
         ↪  collapse = " & "),
        "\\\\"
      ),
      "\\addlinespace",
      # Coefficients for gamma_fea (feature)
      paste(
        "\\multirow{6}{*}{$\\gamma_{\\text{fea}}$} &
         ↪  \\multirow{2}{*}{Regression} &",
        paste(round(coef_mat[7, ], 3), collapse = " & "),
        "\\\\"
```

```
      ),
      paste(
        "& &",
        paste(paste("(", round(coef_mat[8, ], 3), ")", sep = ""),
        ↪  collapse = " & "),
        "\\\\"
      ),
      paste(
        "& \\multirow{2}{*}{2SLS} &",
        paste(round(coef_mat[9, ], 3), collapse = " & "),
        "\\\\"
      ),
      paste(
        "& &",
        paste(paste("(", round(coef_mat[10, ], 3), ")", sep = ""),
        ↪  collapse = " & "),
        "\\\\"
      ),
      paste(
        "& \\multirow{2}{*}{2SLS Corrected} &",
        paste(round(coef_mat[11, ], 3), collapse = " & "),
        "\\\\"
      ),
      paste(
        "& &",
        paste(paste("(", round(coef_mat[12, ], 3), ")", sep = ""),
        ↪  collapse = " & "),
        "\\\\"
      ),
      sep = "\n"
    )
  )
}

# **Prediction Error Analysis**

# If 'data_imprecision' doesn't exist (i.e., we are not doing
↪  robustness simulations),
# perform out-of-sample prediction error analysis.
if (!exists("data_imprecision")) {
  # Number of prediction observations
  n_pred <- 10000
  study <- 4  # Use Study 4 for prediction (most complex case)
  pred_data <- simulate_data(n_pred, study)
```

```r

  # Compute prediction errors for each estimator.
  # Use true regressors (xi) and estimates from Study 4.
  pred_error_ols <- pred_data$y - (coef_mat[1, 4] * pred_data$xi[, 1]
  ↪  + coef_mat[7, 4] * pred_data$xi[, 2])
  pred_error_2sls <- pred_data$y - (coef_mat[3, 4] * pred_data$xi[, 1]
  ↪  + coef_mat[9, 4] * pred_data$xi[, 2])
  pred_error_prop <- pred_data$y - (coef_mat[5, 4] * pred_data$xi[, 1]
  ↪  + coef_mat[11, 4] * pred_data$xi[, 2])

  # Compute standard deviations of the prediction errors
  sd_ols <- sd(pred_error_ols)
  sd_2sls <- sd(pred_error_2sls)
  sd_prop <- sd(pred_error_prop)

  # Compute quantiles of the prediction errors
  probs <- c(0, 0.2, 0.4, 0.5, 0.6, 0.8, 1)
  quantiles_ols <- quantile(pred_error_ols, probs = probs)
  quantiles_2sls <- quantile(pred_error_2sls, probs = probs)
  quantiles_prop <- quantile(pred_error_prop, probs = probs)

  # Prepare data for the table
  estimators <- c("Regression", "2SLS", "Proposed")
  sds <- c(sd_ols, sd_2sls, sd_prop)
  quantiles_list <- list(quantiles_ols, quantiles_2sls, quantiles_prop)

  # Format and print the table rows
  for (i in 1:length(estimators)) {
    estimator_name <- estimators[i]
    quantiles <- quantiles_list[[i]]
    quantiles_fmt <- sprintf("%.4f", quantiles) # Format to 4 decimal
    ↪  places
    sd_fmt <- sprintf("%.4f", sds[i])          # Format to 4 decimal
    ↪  places

    # Create the table row with standard deviations and quantiles
    table_row <- paste(estimator_name, "&", sd_fmt, "&",
    ↪  paste(quantiles_fmt, collapse = " & "), "\\\\")

    # Print the table row
    cat(table_row, "\n")
  }
}
```

F.2 System-of-Equations Model Simulation

Below is a description of how the key constructs in the manuscript and web appendix map to the variables in the system-of-equations simulation R code. This mapping is designed to help readers understand how the theoretical model is implemented in the simulation. Table A9 provides a concise summary.

Notation	Code	Description
y_1, y_2	`y_1`, `y_2`	Dependent variables (e.g., engagement, brand attitude).
z_{nost}	`xi_1[, 1]` (true), `x_1[, 1]` (observed)	Focal AI-coded regressor (e.g., nostalgia) for equation 1.
	`xi_2[, 1]` (true), `x_2[, 1]` (observed)	Focal AI-coded regressor (e.g., nostalgia) for equation 2.
x_{fea}	`xi_1[, 2]` (true), `x_1[, 2]` (observed)	Observed covariate (e.g., product feature) for equation 1.
	`xi_2[, 2]` (true), `x_2[, 2]` (observed)	Observed covariate (e.g., product feature) for equation 2.
w_{her}, w_{cost}	`z_1[, 2]`, `z_1[, 3]`,	Instrumental variables (e.g., heritage, cost). Note
	`z_2[, 2]`, `z_2[, 3]`	that the same instruments are used for both equations.
ε_1, ε_2	`epsilon_1`, `epsilon_2`	Structural error terms for equations 1 and 2.
η_z	`draws[, 10]`	AI-coding error in z_{nost}.
η_w	`draws[, 12]`, `draws[, 13]`	AI-coding error in instruments (Study 4).
ξ_1, ξ_2	`xi_1`, `xi_2`	Matrices of *true* regressors for each equation.
$\hat{\xi}_1$, $\hat{\xi}_2$	`x_1`, `x_2`	Matrices of *observed* regressors for each equation.
ζ_1, ζ_2	`z_1`, `z_2`	Matrices of *true* instruments for each equation.
$\hat{\zeta}_1$, $\hat{\zeta}_2$	`z_1`, `z_2`	Matrices of *observed* instruments for each equation.
ψ_1, ψ_2	`bs_omega_1`, `bs_omega_2`	Cross-product matrices of instruments and coding errors (per equation).
D	`b_3sls_big`	Canonical 3SLS estimator.
D^{con}	`bs_b_3sls_corr[ind, ]`	Bias-corrected 3SLS estimator.
$\widehat{\text{BiasFactor}}_{\text{3SLS}}$	`bs_correction_matrix`	Estimated bias factor.
Σ	`Sigma`	Error covariance matrix (estimated from 2SLS residuals).
Ξ	X	Block-diagonal matrix of *true* regressors.
$\hat{\Xi}$	X	Block-diagonal matrix of *observed* regressors.
$\vec{\zeta}$	Z	Block-diagonal matrix of *true* instruments.
$\vec{\hat{\zeta}}$	Z	Block-diagonal matrix of *observed* instruments.

Table A9: Mapping of Mathematical Notation (Manuscript and Web Appendix) to R Variable Names in the System-of-Equations Simulation Code

The simulation code generates data for a system of two equations. The dependent variables, y_1 (e.g., engagement) and y_2 (e.g., brand attitude), are stored in the code as `y_1` and `y_2`, respectively. The *true* values of the regressors for each equation are stored in the matrices `xi_1` and `xi_2`. The first column of both `xi_1` and `xi_2` (`xi_1[, 1]` and `xi_2[, 1]`) represents the true value of the focal AI-coded regressor, z_{nost} (nostalgia intensity), *before* any endogeneity or measurement error is introduced. The second column (`xi_1[, 2]` and `xi_2[, 2]`) represents the true value of the observed covariate, x_{fea} (product feature). The *observed* regressors, which may include AI-coding error, are stored in `x_1` and `x_2`. The instruments are represented by `z_1` and `z_2`. In studies 1 and 2, `z_1` and `z_2` are identical to `x_1` and `x_2`, respectively. In studies 3 and 4, `z_1` and `z_2` contain the instruments: the first column corresponds to the exogenous component of x_{fea} or x_{fea} itself, the second column (`z_1[, 2]` and `z_2[, 2]`) corresponds to w_{her} (brand heritage), and the third column (`z_1[, 3]` and `z_2[, 3]`) corresponds to w_{cost} (cost of producing nostalgic content). The structural error terms, ε_1 and ε_2, are represented by `epsilon_1` and `epsilon_2`. The AI-coding errors are part of the `draws` matrix: `draws[, 10]` is the error for z_{nost}, and `draws[, 12]` and `draws[, 13]` are the errors for w_{her} and w_{cost}, respectively (in Study 4). The key matrices for bias correction, ψ_1 and ψ_2, are estimated within the bootstrap loop and stored as `bs_omega_1` and `bs_omega_2`. The canonical 3SLS estimator is `b_3sls_big`, and the bias-corrected estimator is `bs_b_3sls_corr[ind, ]`. The simulation constructs stacked matrices X, Z, and Y to represent the system of equations in a compact form for 3SLS estimation. The error covariance matrix, Σ, is estimated using 2SLS residuals and stored as `Sigma`.

```

# Load required library for multivariate normal distributions
library(mvtnorm)

# Clear workspace except for 'sim_ind' and 'data_imprecision'
rm(list = setdiff(ls(), c("sim_ind", "data_imprecision")))

# Ensure 'sim_ind' exists; if not, set it to 1 (default for standard
↪  simulations).
```

```r
if (!exists("sim_ind")) {
  sim_ind <- 1
}

# Check if 'data_imprecision' exists (for robustness analysis)
if (!exists("data_imprecision")) {
  # If 'data_imprecision' doesn't exist, set default simulation parameters
  imprecision <- 0             # Default imprecision level (no added noise)
  study_range <- 1:4           # Studies to run (all four studies)
  n_total <- 50000             # Total number of observations
  n_subsample <- 2500          # Size of the subsample with human-coded data
} else {
  # If 'data_imprecision' exists, use its values (for robustness analysis)
  imprecision <- data_imprecision[sim_ind, "imprecision"] # Get imprecision level
  study_range <- 4             # Only run Study 4 (for robustness analysis)
  n_total <- 50000             # Total number of observations
  n_subsample <- 2500          # Size of the subsample with human-coded data
}

# Function to simulate data based on the specified study scenario
simulate_data <- function(n_total, study) {
  set.seed(102024)             # Set seed for reproducibility

  # Initialize mean vector and covariance matrix for multivariate normal distribution.
  # All variables are initially centered at 0.
  mu <- rep(0, 14)             # Mean vector (zeros for all variables)
  sigma <- diag(14)            # Covariance matrix (identity matrix by default)

  # Variable indices in 'draws' (the matrix of simulated data):
  # [1]  epsilon_1     : Structural error for equation 1 (Engagement)
  # [2]  z_nost_exog   : Exogenous component of z_nost (nostalgia)
  # [3]  x_fea_exog    : Exogenous component of x_fea (feature)
  # [4]  w_her_exog    : Exogenous component of w_her (heritage instrument)
  # [5]  w_cost_exog   : Exogenous component of w_cost (cost instrument)
```

```
# [6]  z_nost_endog  : Endogenous component of z_nost (correlated with epsilon_1 and epsilon_2)
# [7]  (Unused)      : Placeholder.
# [8]  (Unused)      : Placeholder.
# [9]  (Unused)      : Placeholder.
# [10] z_nost_me     : Measurement error in z_nost (AI coding error)
# [11] (Unused)      : Placeholder.
# [12] w_her_me      : Measurement error in w_her (AI coding error)
# [13] w_cost_me     : Measurement error in w_cost (AI coding error)
# [14] epsilon_2     : Structural error for equation 2 (Brand Attitude)

# Structural errors are correlated between equations (key feature of system-of-equations)
sigma[1, 14] <- sigma[14, 1] <- 0.25  # Correlation between epsilon_1 and epsilon_2

# Adjust the covariance matrix 'sigma' based on the study number to introduce
# specific correlations and error structures.
if (study == 1) {
  # Study 1: All regressors are exogenous and measured without error (baseline).
  draws <- rmvnorm(n_total, mean = mu, sigma = sigma)  # Generate data

  # True regressors: z_nost (nostalgia) and x_fea (feature) for both equations.
  xi_1 <- xi_2 <- draws[, c(2, 3)]  # Same regressors for both equations

  # Observed regressors are the same as true regressors (no measurement error).
  x_1 <- xi_1
  x_2 <- xi_2

  # Instruments (zeta) are the same as observed regressors.
  z_1 <- x_1
  z_2 <- x_2

} else if (study == 2) {
  # Study 2: Measurement error in z_nost (nostalgia) correlated with x_fea (feature).
  sigma[3, 10] <- sigma[10, 3] <- 0.4  # Correlation between x_fea_exog and z_nost_me
```

```
76
77      draws <- rmvnorm(n_total, mean = mu, sigma = sigma)  # Generate
        ↪ data
78
79      # True regressors: z_nost (nostalgia) and x_fea (feature).
80      xi_1 <- xi_2 <- draws[, c(2, 3)] # Same regressors for both
        ↪ equations.
81
82      # Observed regressors (x) with measurement error in z_nost.
83      x_1 <- xi_1
84      x_2 <- xi_2
85      x_1[, 1] <- x_1[, 1] + draws[, 10]   # Add measurement error to
        ↪ z_nost in equation 1
86      x_2[, 1] <- x_2[, 1] + draws[, 10]   # Add measurement error to
        ↪ z_nost in equation 2
87
88      # Instruments (zeta) are the same as observed regressors.
89      z_1 <- x_1
90      z_2 <- x_2
91
92    } else if (study == 3) {
93      # Study 3: Endogeneity in z_nost (nostalgia); instruments without
        ↪ measurement error.
94      sigma[3, 10] <- sigma[10, 3] <- 0.4   # Correlation between
        ↪ x_fea_exog and z_nost_me
95      sigma[1, 6] <- sigma[6, 1] <- 0.2     # Correlation between
        ↪ epsilon_1 and z_nost_endog
96      sigma[14, 6] <- sigma[6, 14] <- 0.2   # Correlation between
        ↪ epsilon_2 and z_nost_endog
97      sigma[2, 4] <- sigma[4, 2] <- 0.3     # Correlation between
        ↪ z_nost_exog and w_her_exog
98      sigma[2, 5] <- sigma[5, 2] <- 0.7     # Correlation between
        ↪ z_nost_exog and w_cost_exog
99
100     draws <- rmvnorm(n_total, mean = mu, sigma = sigma)  # Generate
        ↪ data
101
102     # True regressors: z_nost and x_fea, with endogenous component
        ↪ added to z_nost.
103     xi_1 <- xi_2 <- draws[, c(2, 3)] # Same regressors for both
        ↪ equations
104     xi_1[, 1] <- xi_1[, 1] + draws[, 6] # Add endogenous component to
        ↪ z_nost
105     xi_2[, 1] <- xi_2[, 1] + draws[, 6] # Add endogenous component to
        ↪ z_nost
```

```r

    # Observed regressors (x) with measurement error in z_nost.
    x_1 <- xi_1
    x_2 <- xi_2
    x_1[, 1] <- x_1[, 1] + draws[, 10] # Add measurement error to
    ↪ z_nost
    x_2[, 1] <- x_2[, 1] + draws[, 10] # Add measurement error to
    ↪ z_nost

    # Instruments (zeta) are x_fea_exog, w_her_exog, w_cost_exog
    ↪ (without measurement error).
    z_1 <- z_2 <- draws[, c(3, 4, 5)] # Same instruments for both
    ↪ equations

  } else if (study == 4) {
    # Study 4: Endogeneity in z_nost (nostalgia); measurement error in
    ↪ instruments
    # correlated with x_fea (feature).
    sigma[3, 10] <- sigma[10, 3] <- 0.4   # Correlation between
    ↪ x_fea_exog and z_nost_me
    sigma[1, 6] <- sigma[6, 1] <- 0.2     # Correlation between
    ↪ epsilon_1 and z_nost_endog
    sigma[14, 6] <- sigma[6, 14] <- 0.2   # Correlation between
    ↪ epsilon_2 and z_nost_endog
    sigma[2, 4] <- sigma[4, 2] <- 0.3     # Correlation between
    ↪ z_nost_exog and w_her_exog
    sigma[2, 5] <- sigma[5, 2] <- 0.7     # Correlation between
    ↪ z_nost_exog and w_cost_exog
    sigma[3, 12] <- sigma[12, 3] <- 0.3   # Correlation between
    ↪ x_fea_exog and w_her_me
    sigma[3, 13] <- sigma[13, 3] <- 0.3   # Correlation between
    ↪ x_fea_exog and w_cost_me

    draws <- rmvnorm(n_total, mean = mu, sigma = sigma)  # Generate
    ↪ data

    # True regressors: z_nost and x_fea, with endogenous component
    ↪ added to z_nost.
    xi_1 <- xi_2 <- draws[, c(2, 3)] # Same regressors for both
    ↪ equations.
    xi_1[, 1] <- xi_1[, 1] + draws[, 6] # Add endogenous component to
    ↪ z_nost
    xi_2[, 1] <- xi_2[, 1] + draws[, 6] # Add endogenous component to
    ↪ z_nost
```

```r

    # Observed regressors (x) with measurement error in z_nost.
    x_1 <- xi_1
    x_2 <- xi_2
    x_1[, 1] <- x_1[, 1] + draws[, 10] # Add measurement error to
    ↪ z_nost
    x_2[, 1] <- x_2[, 1] + draws[, 10] # Add measurement error to
    ↪ z_nost

    # Instruments (zeta) are x_fea_exog, w_her_exog, w_cost_exog with
    ↪ measurement error.
    z_1 <- z_2 <- draws[, c(3, 4, 5)] # Same instruments for both
    ↪ equations
    z_1[, 2] <- z_1[, 2] + draws[, 12]    # Add measurement error to
    ↪ w_her_exog (heritage)
    z_1[, 3] <- z_1[, 3] + draws[, 13]    # Add measurement error to
    ↪ w_cost_exog (cost)
    z_2[, 2] <- z_2[, 2] + draws[, 12]    # Add measurement error to
    ↪ w_her_exog (heritage)
    z_2[, 3] <- z_2[, 3] + draws[, 13]    # Add measurement error to
    ↪ w_cost_exog (cost)

  } else {
    stop("Study setup not defined") # Error if an invalid study number
    ↪ is provided
  }

  # Structural errors for each equation
  epsilon_1 <- draws[, 1]  # Error for Engagement equation
  epsilon_2 <- draws[, 14] # Error for Brand Attitude equation

  # Dependent variables: y_1 (Engagement) and y_2 (Brand Attitude)
  # Coefficients are implicitly set to 1 for simplicity.
  y_1 <- rowSums(xi_1) + epsilon_1  # Engagement equation
  y_2 <- rowSums(xi_2) + epsilon_2  # Brand Attitude equation

  # Generate coding imprecision deviates (standard normal distribution)
  ↪ for the
  # *manually coded* subsample. Used for SIMEX (robustness analysis).
  imprecision_vec_1 <- rnorm(n_subsample) # For equation 1
  imprecision_vec_2 <- rnorm(n_subsample) # For equation 2

  # Return a list of generated variables
  return(list(
```

```r
    epsilon_1 = epsilon_1,
    epsilon_2 = epsilon_2,
    xi_1 = xi_1,
    xi_2 = xi_2,
    x_1 = x_1,
    x_2 = x_2,
    z_1 = z_1,
    z_2 = z_2,
    y_1 = y_1,
    y_2 = y_2,
    imprecision_vec_1 = imprecision_vec_1,  # For SIMEX
    imprecision_vec_2 = imprecision_vec_2   # For SIMEX
  ))
}

# Initialize matrix to store coefficients and standard errors
# Rows: estimates and standard errors for each coefficient
# Columns: studies 1 to 4
coef_mat <- matrix(0, 16, 4)  # 16 rows: 4 coefficients * (estimate +
 ↪  std. error) * 2 equations

# Loop over the specified studies
for (study in study_range) {
  # Simulate data for the current study
  sim_data <- simulate_data(n_total, study)
  epsilon_1 <- sim_data$epsilon_1
  epsilon_2 <- sim_data$epsilon_2
  xi_1 <- sim_data$xi_1
  xi_2 <- sim_data$xi_2
  x_1 <- sim_data$x_1
  x_2 <- sim_data$x_2
  z_1 <- sim_data$z_1  # Consistently use 'z' for instruments
  z_2 <- sim_data$z_2  # Consistently use 'z' for instruments
  y_1 <- sim_data$y_1
  y_2 <- sim_data$y_2
  imprecision_vec_1 <- sim_data$imprecision_vec_1
  imprecision_vec_2 <- sim_data$imprecision_vec_2

  # **3SLS Estimation**

  # First, perform 2SLS estimation for each equation separately (for
   ↪  initial Sigma estimation)
  b_2sls_1 <- solve(t(x_1) %*% z_1 %*% solve(t(z_1) %*% z_1) %*%
   ↪  t(z_1) %*% x_1) %*%
```

```r
  t(x_1) %*% z_1 %*% solve(t(z_1) %*% z_1) %*% t(z_1) %*% y_1
b_2sls_2 <- solve(t(x_2) %*% z_2 %*% solve(t(z_2) %*% z_2) %*%
 ↪  t(z_2) %*% x_2) %*%
  t(x_2) %*% z_2 %*% solve(t(z_2) %*% z_2) %*% t(z_2) %*% y_2

# Estimate the error covariance matrix 'Sigma' based on 2SLS
 ↪  residuals
Sigma <- matrix(0, 2, 2)
Sigma[1, 1] <- as.numeric(t(y_1 - x_1 %*% b_2sls_1) %*% (y_1 - x_1
 ↪  %*% b_2sls_1) / (n_total - ncol(x_1)))
Sigma[1, 2] <- Sigma[2, 1] <- as.numeric(t(y_1 - x_1 %*% b_2sls_1)
 ↪  %*% (y_2 - x_2 %*% b_2sls_2) / (n_total - ncol(x_1)))
Sigma[2, 2] <- as.numeric(t(y_2 - x_2 %*% b_2sls_2) %*% (y_2 - x_2
 ↪  %*% b_2sls_2) / (n_total - ncol(x_2)))

# Stack the X matrices for both equations
k1 <- ncol(x_1)  # Number of regressors in equation 1
k2 <- ncol(x_2)  # Number of regressors in equation 2
X <- matrix(0, n_total * 2, k1 + k2)
X[1:n_total, 1:k1] <- x_1
X[(n_total + 1):(2 * n_total), (k1 + 1):(k1 + k2)] <- x_2

# Stack the Z matrices for both equations
p1 <- ncol(z_1)  # Number of instruments in equation 1
p2 <- ncol(z_2)  # Number of instruments in equation 2
Z <- matrix(0, n_total * 2, p1 + p2)
Z[1:n_total, 1:p1] <- z_1
Z[(n_total + 1):(2 * n_total), (p1 + 1):(p1 + p2)] <- z_2

# Construct the big Sigma matrix (variance-covariance matrix of the
 ↪  system)
big_Sigma <- matrix(0, p1 + p2, p1 + p2)
big_Sigma[1:p1, 1:p1] <- Sigma[1, 1] * t(z_1) %*% z_1
big_Sigma[1:p1, (p1 + 1):(p1 + p2)] <- Sigma[1, 2] * t(z_1) %*% z_2
big_Sigma[(p1 + 1):(p1 + p2), 1:p1] <- Sigma[2, 1] * t(z_2) %*% z_1
big_Sigma[(p1 + 1):(p1 + p2), (p1 + 1):(p1 + p2)] <- Sigma[2, 2] *
 ↪  t(z_2) %*% z_2

# Stack the Y vectors for both equations
Y <- c(y_1, y_2)

# **3SLS Estimator**
b_3sls <- solve(t(X) %*% Z %*% solve(big_Sigma) %*% t(Z) %*% X) %*%
  t(X) %*% Z %*% solve(big_Sigma) %*% t(Z) %*% Y
```

```r

  # Compute standard errors of the 3SLS estimator
  b_3sls_sd <- sqrt(diag(solve(t(X) %*% Z %*% solve(big_Sigma) %*%
  ↪  t(Z) %*% X)))

  # Store 3SLS estimates and standard errors (initial estimates without
  ↪  correction)
  coef_mat[1, study] <- b_3sls[1]           # beta_eng,nost estimate
  coef_mat[2, study] <- b_3sls_sd[1]        # beta_eng,nost standard
  ↪  error
  coef_mat[5, study] <- b_3sls[2]           # gamma_eng,fea estimate
  coef_mat[6, study] <- b_3sls_sd[2]        # gamma_eng,fea standard
  ↪  error
  coef_mat[9, study] <- b_3sls[k1 + 1]      # beta_brand,nost estimate
  coef_mat[10, study] <- b_3sls_sd[k1 + 1]  # beta_brand,nost standard
  ↪  error
  coef_mat[13, study] <- b_3sls[k1 + 2]     # gamma_brand,fea estimate
  coef_mat[14, study] <- b_3sls_sd[k1 + 2]  # gamma_brand,fea standard
  ↪  error

  # **Bootstrap Procedure for Corrected 3SLS Estimates**

  # Initialize matrix to store bootstrap corrected estimates
  bs_b_3sls_corr <- matrix(0, 1000, k1 + k2)

  for (ind in 1:1000) {
    # Bootstrap sample indices for the subsample with human-coded data
    subsample_indices <- sample(1:n_subsample, n_subsample, replace =
    ↪  TRUE)

    # Introduce coding imprecision in the subsample for both equations
    ↪  (for SIMEX)
    xi_imprecise_subsample_1 <- xi_1[subsample_indices, ]
    xi_imprecise_subsample_1[, 1] <- xi_imprecise_subsample_1[, 1] +
    ↪  imprecision_vec_1 * imprecision # Add noise to z_nost
    xi_imprecise_subsample_2 <- xi_2[subsample_indices, ]
    xi_imprecise_subsample_2[, 1] <- xi_imprecise_subsample_2[, 1] +
    ↪  imprecision_vec_2 * imprecision # Add noise to z_nost

    # Calculate Omega matrices for each equation (key for bias
    ↪  correction).
    # These matrices capture the relationship between the instruments
    ↪  and the
    # AI coding errors, estimated using the calibration subsample.
```

```r
    bs_omega_1 <- t(z_1[subsample_indices, ]) %*%
    ↪  (x_1[subsample_indices, ] - xi_imprecise_subsample_1) /
    ↪  n_subsample
    bs_omega_2 <- t(z_2[subsample_indices, ]) %*%
    ↪  (x_2[subsample_indices, ] - xi_imprecise_subsample_2) /
    ↪  n_subsample

    # Stack Omega matrices
    bs_Omega <- matrix(0, p1 + p2, k1 + k2)
    bs_Omega[1:p1, 1:k1] <- bs_omega_1
    bs_Omega[(p1 + 1):(p1 + p2), (k1 + 1):(k1 + k2)] <- bs_omega_2

    # Bootstrap sample indices for the larger dataset (the remaining
    ↪  data)
    total_indices <- sample((n_subsample + 1):n_total, n_total -
    ↪  n_subsample, replace = TRUE)

    # Extract bootstrap samples for each equation
    big_x_1 <- x_1[total_indices, ]
    big_z_1 <- z_1[total_indices, ]
    big_y_1 <- y_1[total_indices]
    big_x_2 <- x_2[total_indices, ]
    big_z_2 <- z_2[total_indices, ]
    big_y_2 <- y_2[total_indices]

    # Stack the X, Z, and Y matrices for the bootstrap sample
    n_bootstrap <- length(total_indices)
    X_big <- matrix(0, n_bootstrap * 2, k1 + k2)
    X_big[1:n_bootstrap, 1:k1] <- big_x_1
    X_big[(n_bootstrap + 1):(2 * n_bootstrap), (k1 + 1):(k1 + k2)] <-
    ↪  big_x_2

    Z_big <- matrix(0, n_bootstrap * 2, p1 + p2)
    Z_big[1:n_bootstrap, 1:p1] <- big_z_1
    Z_big[(n_bootstrap + 1):(2 * n_bootstrap), (p1 + 1):(p1 + p2)] <-
    ↪  big_z_2

    Y_big <- c(big_y_1, big_y_2)

    # Estimate error covariance matrix bs_Sigma for the bootstrap
    ↪  sample (using 2SLS residuals)
    bs_b_2sls_1 <- solve(t(big_x_1) %*% big_z_1 %*% solve(t(big_z_1)
    ↪  %*% big_z_1) %*% t(big_z_1) %*% big_x_1) %*%
      t(big_x_1) %*% big_z_1 %*% solve(t(big_z_1) %*% big_z_1) %*%
      ↪  t(big_z_1) %*% big_y_1
```

```
    bs_b_2sls_2 <- solve(t(big_x_2) %*% big_z_2 %*% solve(t(big_z_2) %*% big_z_2) %*% t(big_z_2) %*% big_x_2) %*%
      t(big_x_2) %*% big_z_2 %*% solve(t(big_z_2) %*% big_z_2) %*% t(big_z_2) %*% big_y_2

    bs_Sigma <- matrix(0, 2, 2)
    bs_Sigma[1, 1] <- as.numeric(t(big_y_1 - big_x_1 %*% bs_b_2sls_1) %*% (big_y_1 - big_x_1 %*% bs_b_2sls_1) / (n_bootstrap - ncol(big_x_1)))
    bs_Sigma[1, 2] <- bs_Sigma[2, 1] <- as.numeric(t(big_y_1 - big_x_1 %*% bs_b_2sls_1) %*% (big_y_2 - big_x_2 %*% bs_b_2sls_2) / (n_bootstrap - ncol(big_x_1)))
    bs_Sigma[2, 2] <- as.numeric(t(big_y_2 - big_x_2 %*% bs_b_2sls_2) %*% (big_y_2 - big_x_2 %*% bs_b_2sls_2) / (n_bootstrap - ncol(big_x_2)))

    # Construct bs_big_Sigma matrix for the bootstrap sample
    bs_big_Sigma <- matrix(0, p1 + p2, p1 + p2)
    bs_big_Sigma[1:p1, 1:p1] <- bs_Sigma[1, 1] * t(big_z_1) %*% big_z_1
    bs_big_Sigma[1:p1, (p1 + 1):(p1 + p2)] <- bs_Sigma[1, 2] * t(big_z_1) %*% big_z_2
    bs_big_Sigma[(p1 + 1):(p1 + p2), 1:p1] <- bs_Sigma[2, 1] * t(big_z_2) %*% big_z_1
    bs_big_Sigma[(p1 + 1):(p1 + p2), (p1 + 1):(p1 + p2)] <- bs_Sigma[2, 2] * t(big_z_2) %*% big_z_2

    # 3SLS estimation on the bootstrap sample. This corresponds to 'D' in Equation
    # (25) of the Web Appendix.
    b_3sls_big <- solve(t(X_big) %*% Z_big %*% solve(bs_big_Sigma) %*% t(Z_big) %*% X_big) %*%
      t(X_big) %*% Z_big %*% solve(bs_big_Sigma) %*% t(Z_big) %*% Y_big

    # Correction matrix for the estimator. This uses bs_Omega (estimated from the
    # calibration subsample) to correct for the bias in the 3SLS estimator.
    # This corresponds to  (I_{Gg} - \widehat{BiasFactor}_{3SLS}) in Equation (25).
    bs_correction_matrix <- diag(k1 + k2) - n_bootstrap * solve(
      t(X_big) %*% Z_big %*% solve(bs_big_Sigma) %*% t(Z_big) %*% X_big
    ) %*% t(X_big) %*% Z_big %*% solve(bs_big_Sigma) %*% bs_Omega

    # Corrected 3SLS estimates. This corresponds to D^{con} in Equation (25).
```

```r
    bs_b_3sls_corr[ind, ] <- solve(bs_correction_matrix) %*% b_3sls_big
  }

  # Compute standard deviations and medians from bootstrap corrected estimates
  b_3sls_corr_sd <- apply(bs_b_3sls_corr, 2, sd)
  b_3sls_corr_median <- apply(bs_b_3sls_corr, 2, median)

  # Store corrected 3SLS estimates and standard errors
  coef_mat[3, study] <- b_3sls_corr_median[1]        # beta_eng,nost corrected estimate
  coef_mat[4, study] <- b_3sls_corr_sd[1]            # beta_eng,nost standard error
  coef_mat[7, study] <- b_3sls_corr_median[2]        # gamma_eng,fea corrected estimate
  coef_mat[8, study] <- b_3sls_corr_sd[2]            # gamma_eng,fea standard error
  coef_mat[11, study] <- b_3sls_corr_median[3]       # beta_brand,nost corrected estimate
  coef_mat[12, study] <- b_3sls_corr_sd[3]           # beta_brand,nost standard error
  coef_mat[15, study] <- b_3sls_corr_median[4]       # gamma_brand,fea corrected estimate
  coef_mat[16, study] <- b_3sls_corr_sd[4]           # gamma_brand,fea standard error
}

# Output the results in LaTeX table format if 'data_imprecision' doesn't exist
if (!exists("data_imprecision")) {
  cat(
    paste(
      # beta_eng,nost estimates (Engagement - Nostalgia)
      paste(
        "\\multirow{4}{*}{$\\beta_{\\text{eng, nost}}$} & \\multirow{2}{*}{3SLS} &",
        paste(round(coef_mat[1, ], 3), collapse = " & "),
        "\\\\"
      ),
      paste(
        "& & ",
        paste(paste("(", round(coef_mat[2, ], 3), ")", sep = ""), collapse = " & "),
        "\\\\"
```

```
      ),
      paste(
        "& \\multirow{2}{*}{3SLS Corrected} &",
        paste(round(coef_mat[3, ], 3), collapse = " & "),
        "\\\\"
      ),
      paste(
        "& & ",
        paste(paste("(", round(coef_mat[4, ], 3), ")", sep = ""),
        ↪  collapse = " & "),
        "\\\\"
      ),
      "\\addlinespace",
      # gamma_eng,fea estimates (Engagement - Feature)
      paste(
        "\\multirow{4}{*}{$\\gamma_{\\text{eng, fea}}$} &
        ↪  \\multirow{2}{*}{3SLS} &",
        paste(round(coef_mat[5, ], 3), collapse = " & "),
        "\\\\"
      ),
      paste(
        "& & ",
        paste(paste("(", round(coef_mat[6, ], 3), ")", sep = ""),
        ↪  collapse = " & "),
        "\\\\"
      ),
      paste(
        "& \\multirow{2}{*}{3SLS Corrected} &",
        paste(round(coef_mat[7, ], 3), collapse = " & "),
        "\\\\"
      ),
      paste(
        "& & ",
        paste(paste("(", round(coef_mat[8, ], 3), ")", sep = ""),
        ↪  collapse = " & "),
        "\\\\"
      ),
      "\\addlinespace",
      # beta_brand,nost estimates (Brand Attitude - Nostalgia)
      paste(
        "\\multirow{4}{*}{$\\beta_{\\text{brand, nost}}$} &
        ↪  \\multirow{2}{*}{3SLS} &",
        paste(round(coef_mat[9, ], 3), collapse = " & "),
        "\\\\"
```

```
      ),
      paste(
        "& & ",
        paste(paste("(", round(coef_mat[10, ], 3), ")", sep = ""),
        ↪  collapse = " & "),
        "\\\\"
      ),
      paste(
        "& \\multirow{2}{*}{3SLS Corrected} &",
        paste(round(coef_mat[11, ], 3), collapse = " & "),
        "\\\\"
      ),
      paste(
        "& & ",
        paste(paste("(", round(coef_mat[12, ], 3), ")", sep = ""),
        ↪  collapse = " & "),
        "\\\\"
      ),
      "\\addlinespace",
      # gamma_brand,fea estimates (Brand Attitude - Feature)
      paste(
        "\\multirow{4}{*}{$\\gamma_{\\text{brand, fea}}$} &
        ↪  \\multirow{2}{*}{3SLS} &",
        paste(round(coef_mat[13, ], 3), collapse = " & "),
        "\\\\"
      ),
      paste(
        "& & ",
        paste(paste("(", round(coef_mat[14, ], 3), ")", sep = ""),
        ↪  collapse = " & "),
        "\\\\"
      ),
      paste(
        "& \\multirow{2}{*}{3SLS Corrected} &",
        paste(round(coef_mat[15, ], 3), collapse = " & "),
        "\\\\"
      ),
      paste(
        "& & ",
        paste(paste("(", round(coef_mat[16, ], 3), ")", sep = ""),
        ↪  collapse = " & "),
        "\\\\"
      ),
      sep = "\n"
```

```
    )
  )
}
```

F.3 Robustness and Extrapolation Simulations

This final section of the simulation code assesses the robustness of the proposed estimators to imprecision in the manually coded calibration data and implements the Simulation-Extrapolation (SIMEX) method to correct for this imprecision. The code first initializes a data frame, `data_imprecision`, to store the results. This data frame includes a column, `imprecision`, which holds a sequence of values representing increasing levels of artificially added noise to the manually coded variables. These values range from 0 to 3, representing the standard deviation of the added noise as a multiple of the estimated standard deviation of the *original* manual coding error (as described in the Implementation Guide).

The core of the robustness analysis is a loop (controlled by `sim_ind`) that iterates over the different `imprecision` levels. For each level, the code executes the compiled single-equation and system-of-equations simulation scripts (using `loadcmp` with the `.Rc` files). These scripts, as described previously, generate data, estimate the model parameters (including the bias-corrected estimators), and return the results. The key results – the bias-corrected estimates from the single-equation and system-of-equations models – are stored in the `data_imprecision` data frame in the `single_proposed` and `system_proposed` columns, respectively. This creates a dataset where each row corresponds to a specific level of added noise, and the columns hold the resulting coefficient estimates.

To visualize the impact of manual coding imprecision, the code uses `ggplot2` to plot the `single_proposed` and `system_proposed` estimates against the `imprecision` levels. This plot shows how the estimated coefficients change as the manually coded data becomes increasingly noisy.

Finally, the code implements the SIMEX extrapolation. It fits a Generalized Additive Model

(GAM) with a cubic spline (`s(x, bs = "cs")`) to the relationship between the estimated coefficients (`single_proposed` and `system_proposed`) and the `imprecision` levels. The `predict` function is then used to extrapolate the fitted GAM back to the case of zero added imprecision (`x = 0`). These extrapolated estimates, representing the corrected coefficients, are stored in the `single_pred` and `system_pred` columns of the `data_imprecision` data frame. A second plot visualizes these extrapolated estimates, demonstrating the effectiveness of the SIMEX correction. The entire process, including the loop over imprecision levels and the GAM fitting/extrapolation, is embedded within the bootstrap procedure of the main simulation scripts (as described in the Implementation Guide), allowing for the calculation of standard errors and confidence intervals for the SIMEX-corrected estimates.

```

# Load required libraries
library(compiler)     # For compiling R scripts to byte code (for speed)
library(dplyr)        # For data manipulation (using pipes: %>%)
library(gam)          # For Generalized Additive Models (for extrapolation)
library(ggplot2)      # For plotting
library(mgcv)         # For GAMs (an extension of 'gam' package, often provides more options)
library(mvtnorm)      # For multivariate normal distributions (used in data generation)
library(tidyr)        # For data reshaping (pivot_longer)

# Clear workspace and set seed for reproducibility
rm(list = ls())
set.seed(2025)         # Set a *different* seed than the data generation for independent simulations
enableJIT(3)           # Enable Just-In-Time compilation (level 3 for maximum optimization)

# Initialize data frame to store results
data_imprecision <- data.frame(
  imprecision = 0:300 / 100,      # Imprecision levels from 0 to 3.0 in steps of 0.01
  single_proposed = rep(0, 301),  # Placeholder for single-equation model estimates
  system_proposed = rep(0, 301)   # Placeholder for system-of-equations model estimates
```

```r
)

# Compile the simulation scripts for efficiency.  CRUCIAL: Use relative paths or
# a robust method to locate these files.  Absolute paths will NOT work on other systems.
# Assuming the simulation scripts are in a "simulation_scripts" subdirectory:
cmpfile(
  file.path("simulation_scripts", "Sim Single 2024_10_15.R"),
  options = list(optimize = 3)
)
cmpfile(
  file.path("simulation_scripts", "Sim System 2024_10_15.R"),
  options = list(optimize = 3)
)

# UNCOMMENT BELOW TO REGENERATE RAW RESULTS (this takes a long time!)
# # Loop over different levels of imprecision (this is the main SIMEX loop)
# for (sim_ind in 1:301) {
#
#   print(sim_ind)  # Print current simulation index (for progress tracking)
#
#   # Load and execute the compiled single-equation model simulation.
#   # The 'sim_ind' and 'data_imprecision' variables will be available to the
#   # loaded script, controlling the simulation parameters.
#   loadcmp(file.path("simulation_scripts", "Sim Single 2024_10_15.Rc"))
#
#   # Store the proposed estimator from the single-equation model
#   data_imprecision$single_proposed[sim_ind] <- coef_mat[5, 4] # 2SLS corrected estimate, study 4
#
#   # Load and execute the compiled system-of-equations model simulation
#   loadcmp(file.path("simulation_scripts", "Sim System 2024_10_15.Rc"))
#
#   # Store the proposed estimator from the system-of-equations model
#   data_imprecision$system_proposed[sim_ind] <- coef_mat[3, 4] # 3SLS corrected estimate, study 4, first coefficient
```

```r
#
# }
#
# # Save the results (use a relative path!)
# save(data_imprecision, file = file.path("simulation_results",
↪  "Robustness_Results_2024_10_15.Rdata"))


# Load the results (comment out the loop above and uncomment this to
↪  use pre-saved results)
#  Make sure the path is correct for your system.
load(file.path("simulation_results",
↪  "Robustness_Results_2024_10_15.Rdata"))


# Prepare data for plotting by reshaping to long format (for ggplot2)
data_to_plot <- tidyr::pivot_longer(
  data_imprecision,
  cols = c("single_proposed", "system_proposed"),
  names_to = "Model",
  values_to = "Estimate"
)

# Update model names for clarity in the plot
data_to_plot$Model[data_to_plot$Model == "single_proposed"] <-
↪  "Single-Equation Model"
data_to_plot$Model[data_to_plot$Model == "system_proposed"] <-
↪  "System-of-Equations Model"

# Plot the estimates against imprecision levels (from 0 to 2.0)
ggplot(data_to_plot %>% filter(imprecision >= 0 & imprecision <= 2),
       aes(x = imprecision, y = Estimate)) +
  geom_point() +  # Scatter plot of the raw estimates
  geom_smooth(method = "gam") +  # Add a smoothed line using GAM
  xlab("Imprecision in Manually-Coded Measure (% of Standard Deviation
  ↪  of Manually-Coded Variable)") + # Clear x-axis label
  ylab("Estimated Coefficient") +
  scale_x_continuous(labels = scales::percent) + # Format x-axis as
  ↪  percentages
  facet_wrap(~ Model, nrow = 2, scales = "free_y") + # Separate plots
  ↪  for each model
  theme_bw() # Use a black-and-white theme

# Save the plot (use a relative path!)
```

```r
ggsave(
  filename = "Simulations_Robustness.png",
  plot = last_plot(),
  width = 7,
  height = 5,
  units = "in",
  dpi = 300,
  path = "figures"  # Save in a "figures" subdirectory
)

# Initialize columns for extrapolated estimates
data_imprecision$single_pred <- 0
data_imprecision$system_pred <- 0

# Extrapolate to imprecision = 0 using GAM (Generalized Additive Models).
# This is the core of the SIMEX method. We're fitting a smooth curve to the
# relationship between the estimated coefficient and the level of added noise,
# and then extrapolating that curve back to the point where there's *no* added noise.
for (ind in 2:251) { # Start at 2 because we need some data points to fit the GAM

  # Fit GAM on data from current imprecision level to the maximum and predict at imprecision = 0.
  # We use a cubic spline (bs = "cs") for flexibility.
  data_imprecision$single_pred[ind] <- predict(
    gam(y ~ s(x, bs = "cs"),  # Fit a GAM with a cubic spline smooth for x
        data = data.frame(
          y = data_imprecision$single_proposed[ind:301], # Use data *from* the current imprecision level onwards
          x = ((ind - 1):300) / 100  # x values are the imprecision levels (0 to 3)
        )),
    newdata = data.frame(x = 0) # Predict at x = 0 (no added imprecision)
  )

  data_imprecision$system_pred[ind] <- predict(
    gam(y ~ s(x, bs = "cs"),
        data = data.frame(
```

```
            y = data_imprecision$system_proposed[ind:301], # Use data
             ↪ *from* the current imprecision level onwards
            x = ((ind - 1):300) / 100  # x values are the imprecision
             ↪ levels
        )),
    newdata = data.frame(x = 0) # Predict at x = 0
  )
}

# Prepare data for plotting by reshaping to long format
data_to_plot <- tidyr::pivot_longer(
  data_imprecision,
  cols = c("single_pred", "system_pred"), # Now using the
   ↪ *extrapolated* estimates
  names_to = "Model",
  values_to = "Estimate"
)

# Update model names for clarity
data_to_plot$Model[data_to_plot$Model == "single_pred"] <-
 ↪ "Single-Equation Model"
data_to_plot$Model[data_to_plot$Model == "system_pred"] <-
 ↪ "System-of-Equations Model"

# Plot the *extrapolated* estimates against imprecision levels (from 0
 ↪ to 2.0)
ggplot(data_to_plot %>% filter(imprecision > 0 & imprecision <= 2), #
 ↪ Filter for relevant range
       aes(x = imprecision, y = Estimate)) +
  geom_point() +  # Scatter plot of the extrapolated estimates
  geom_smooth(method = "gam") +  # Add a smoothed line
  xlab("Imprecision in Manually-Coded Measure (% of Standard Deviation
   ↪ of Manually-Coded Variable)") + # Clear x-axis label
  ylab("Extrapolated Coefficient Estimate") + # Clear y-axis label
  scale_x_continuous(labels = scales::percent) + # Format x-axis as
   ↪ percentages
  facet_wrap(~ Model, nrow = 2, scales = "free_y") + # Separate plots
   ↪ for each model
  theme_bw()

# Save the extrapolation plot (use a relative path!)
ggsave(
  filename = "Simulations_Extrapolation.png",
  plot = last_plot(),
```

```
  width = 7,
  height = 5,
  units = "in",
  dpi = 300,
  path = "figures"  # Save in a "figures" subdirectory
)
```

G Bibliography

Aagerup, Ulf (2011), "The influence of real women in advertising on mass market fashion brand perception," *Journal of Fashion Marketing and Management: An International Journal*, 15 (4), 486–502.

Abel, Andrew B (2017), "Classical measurement error with several regressors," *Wharton School of the University of Pennsylvania*.

Alba, J (1999), "President's column: Loose ends," *Assoc. Consum. Res. Newsl*, 2.

Alden, Dana L, Wayne D Hoyer, and Chol Lee (1993), "Identifying global and culture-specific dimensions of humor in advertising: A multinational analysis," *Journal of Marketing*, 57 (2), 64–75.

Aydemir, Abdurrahman and George J Borjas (2011), "Attenuation bias in measuring the wage impact of immigration," *Journal of Labor Economics*, 29 (1), 69–112.

Baker, Michael J and Gilbert A Churchill Jr (1977), "The impact of physically attractive models on advertising evaluations," *Journal of Marketing research*, 14 (4), 538–55.

Bakhshi, Saeideh, David A Shamma, and Eric Gilbert (2014), "Faces engage us: Photos with faces attract more likes and comments on instagram," in *Proceedings of the SIGCHI conference on human factors in computing systems*, 965–74.

Bao, Yang and Anindya Datta (2014), "Simultaneously discovering and quantifying risk types from textual risk disclosures," *Management Science*, 60 (6), 1371–91.

Beard, Fred K (2005), "One hundred years of humor in american advertising," *Journal of Macromarketing*, 25 (1), 54–65.

Belk, Russell W (2013), "Extended self in a digital world," *Journal of consumer research*, 40 (3), 477–500.

Berger, Jonah and Katherine L Milkman (2012), "What makes online content viral?" *Journal of marketing research*, 49 (2), 192–205.

Bound, John, Charles Brown, and Nancy Mathiowetz (2001), "Measurement error in survey data," in *Handbook of econometrics*, Elsevier, 3705–3843.

Buolamwini, Joy and Timnit Gebru (2018), "Gender shades: Intersectional accuracy disparities in commercial gender classification," in *Conference on fairness, accountability and transparency*, PMLR, 77–91.

Carroll, Raymond J, David Ruppert, Leonard A Stefanski, and Ciprian M Crainiceanu (2006), *Measurement error in nonlinear models: A modern perspective*, Chapman; Hall/CRC.

Chattopadhyay, Amitava and Kunal Basu (1990), "Humor in advertising: The moderating role of prior brand evaluation," *Journal of Marketing Research*, 27 (4), 466–76.

Chen, Xiaohong, Han Hong, and Elie Tamer (2005), "Measurement error models with auxiliary data," *The Review of Economic Studies*, 72 (2), 343–66.

Chung, Hwiman and Xinshu Zhao (2003), "Humour effect on memory and attitude: Moderating role of product involvement," *International Journal of Advertising*, 22 (1), 117–44.

Cook, John R and Leonard A Stefanski (1994), "Simulation-extrapolation estimation in parametric measurement error models," *Journal of the American Statistical association*, 89 (428), 1314–28.

Cunha, Flavio, James J Heckman, and Susanne M Schennach (2010), "Estimating the technology of

cognitive and noncognitive skill formation," *Econometrica*, 78 (3), 883–931.
Dahl, Darren W, Jaideep Sengupta, and Kathleen D Vohs (2009), "Sex in advertising: Gender differences and the role of relationship commitment," *Journal of Consumer Research*, 36 (2), 215–31.
De Vries, Lisette, Sonja Gensler, and Peter SH Leeflang (2012), "Popularity of brand posts on brand fan pages: An investigation of the effects of social media marketing," *Journal of interactive marketing*, 26 (2), 83–91.
Egami, Naoki, Christian J Fong, Justin Grimmer, Margaret E Roberts, and Brandon M Stewart (2018), "How to make causal inferences using texts," *arXiv preprint arXiv:1802.02163*.
Eisend, Martin (2009), "A meta-analysis of humor in advertising," *Journal of the Academy of Marketing Science*, 37, 191–203.
Feurer, Matthias, Aaron Klein, Katharina Eggensperger, Jost Springenberg, Manuel Blum, and Frank Hutter (2015), "Efficient and robust automated machine learning," *Advances in neural information processing systems*, 28.
Fuller, Wayne A (2009), *Measurement error models*, John Wiley & Sons.
Garcia-Rada, Ximena, Michael I Norton, and Rebecca K Ratner (2024), "A desire to create shared memories increases consumers' willingness to sacrifice experience quality for togetherness," *Journal of Consumer Psychology*, 34 (2), 247–63.
Gordon, Brett R, Florian Zettelmeyer, Neha Bhargava, and Dan Chapsky (2019), "A comparison of approaches to advertising measurement: Evidence from big field experiments at facebook," *Marketing Science*, 38 (2), 193–225.
Grewal, Rajdeep and Yeşim Orhun (2024), "Unpacking the instrumental variables approach," *Impact at JMR*.
Gulas, Charles S and Marc G Weinberger (2006), *Humor in advertising: A comprehensive analysis*, ME Sharpe.
Gustafson, Paul (2003), *Measurement error and misclassification in statistics and epidemiology: Impacts and bayesian adjustments*, Chapman; Hall/CRC.
Hamilton, Rebecca W (2024), "Leveraging opportunities and managing risks in marketing research," *Journal of Marketing Research*, SAGE Publications Sage CA: Los Angeles, CA.
Hausman, Jerry (2001), "Mismeasured variables in econometric analysis: Problems from the right and problems from the left," *Journal of Economic perspectives*, 15 (4), 57–67.
He, Xin, Kaiyong Zhao, and Xiaowen Chu (2021), "AutoML: A survey of the state-of-the-art," *Knowledge-Based Systems*, 212, 106622.
Hirsch, Alan R (1992), "Nostalgia: A neuropsychiatrie understanding." *Advances in consumer research*, 19 (1).
Holak, Susan L and William J Havlena (1998), "Feelings, fantasies, and memories: An examination of the emotional components of nostalgia," *Journal of business research*, 42 (3), 217–26.
Holbrook, Morris B and Robert M Schindler (1991), "Echoes of the dear departed past: Some work in progress on nostalgia." *Advances in consumer research*, 18 (1).
Hu, Yingyao (2008), "Identification and estimation of nonlinear models with misclassification error using instrumental variables: A general solution," *Journal of Econometrics*, 144 (1), 27–61.
Hussain, Zaeem, Mingda Zhang, Xiaozhong Zhang, Keren Ye, Christopher Thomas, Zuha Agha, Nathan Ong, and Adriana Kovashka (2017), "Automatic understanding of image and video advertisements," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 1705–15.

Hutter, Frank, Lars Kotthoff, and Joaquin Vanschoren (2019), *Automated machine learning: Methods, systems, challenges*, Springer Nature.

Jones, Polly Wyn (2023), "Starting to laugh again. The slow return of humour in advertising."

Kim, Seungbae, Jyun-Yu Jiang, and Wei Wang (2021), "Discovering undisclosed paid partnership on social media via aspect-attentive sponsored post learning," in *Proceedings of the 14th ACM international conference on web search and data mining*, 319–27.

Lamberton, Cait and Andrew T Stephen (2016), "A thematic exploration of digital, social media, and mobile marketing: Research evolution from 2000 to 2015 and an agenda for future inquiry," *Journal of marketing*, 80 (6), 146–72.

Lang, Peter J and Margaret M Bradley (2010), "Emotion and the motivational brain," *Biological psychology*, 84 (3), 437–50.

Lasaleta, Jannine D, Constantine Sedikides, and Kathleen D Vohs (2014), "Nostalgia weakens the desire for money," *Journal of Consumer Research*, 41 (3), 713–29.

Lee, Dokyun, Kartik Hosanagar, Harikesh Nair, and others (2013), "The effect of advertising content on consumer engagement: Evidence from facebook," Working paper.

Loveland, Katherine E, Dirk Smeesters, and Naomi Mandel (2010), "Still preoccupied with 1995: The need to belong and preference for nostalgic products," *Journal of Consumer Research*, 37 (3), 393–408.

Lull, Robert B and Brad J Bushman (2015), "Do sex and violence sell? A meta-analytic review of the effects of sexual and violent media and ad content on memory, attitudes, and buying intentions." *Psychological bulletin*, 141 (5), 1022.

Lynch Jr, John G (1982), "On the external validity of experiments in consumer research," *Journal of consumer Research*, 9 (3), 225–39.

Media, Kantar (2022), "Who's laughing now? Let's stop the decline of humour in advertising."

Meyer, Robert J (2015), "A field guide to publishing in an era of doubt," *Journal of Marketing Research*, SAGE Publications Sage CA: Los Angeles, CA.

Muehling, Darrel D, David E Sprott, and Abdullah J Sultan (2014), "Exploring the boundaries of nostalgic advertising effects: A consideration of childhood brand exposure and attachment on consumers' responses to nostalgia-themed advertisements," *Journal of Advertising*, 43 (1), 73–84.

Mukherjee, Anirban and Vrinda Kadiyali (2011), "Modeling multichannel home video demand in the US motion picture industry," *Journal of Marketing Research*, 48 (6), 985–95.

Olney, Thomas J, Morris B Holbrook, and Rajeev Batra (1991), "Consumer responses to advertising: The effects of ad content, emotions, and attitude toward the ad on viewing time," *Journal of consumer research*, 17 (4), 440–53.

Ordabayeva, Nailya, Monika Lisjak, and Aziza C Jones (2022), "How social perceptions influence consumption for self, for others, and within the broader system," *Current Opinion in Psychology*, 43, 30–35.

Pentina, Iryna, Véronique Guilloux, and Anca Cristina Micu (2018), "Exploring social media engagement behaviors in the context of luxury brands," *Journal of Advertising*, 47 (1), 55–69.

Petty, Richard E, Duane T Wegener, and Leandre R Fabrigar (1997), "Attitudes and attitude change," *Annual review of psychology*, 48 (1), 609–47.

Pham, Michel Tuan, Maggie Geuens, and Patrick De Pelsmacker (2013), "The influence of ad-evoked feelings on brand evaluations: Empirical generalizations from consumer responses to more than 1000 TV commercials," *International Journal of Research in Marketing*, 30 (4), 383–94.

Reichert, Tom (2002), "Sex in advertising research: A review of content, effects, and functions of sexual information in consumer advertising," *Annual review of sex research*, 13 (1), 241–73.

Ritzer, George, Paul Dean, and Nathan Jurgenson (2012), "The coming of age of the prosumer," *American behavioral scientist*, 56 (4), 379–98.

Schennach, Susanne M (2004), "Estimation of nonlinear models with measurement error," *Econometrica*, 72 (1), 33–75.

Schennach, Susanne M (2016), "Recent advances in the measurement error literature," *Annual Review of Economics*, 8 (1), 341–77.

Seijo, Emilio and Bodhisattva Sen (2011), "A continuous mapping theorem for the smallest argmax functional."

Sengupta, Jaideep and Darren W Dahl (2008), "Gender-related reactions to gratuitous sex appeals in advertising," *Journal of consumer psychology*, 18 (1), 62–78.

Sepanski, Jungsywan H and Raymond J Carroll (1993), "Semiparametric quasilikelihood and variance function estimation in measurement error models," *Journal of Econometrics*, 58 (1-2), 223–56.

Shanks, David R, Miguel A Vadillo, Benjamin Riedel, Ashley Clymo, Sinita Govind, Nisha Hickin, Amanda JF Tamman, and Lara Puhlmann (2015), "Romance, risk, and replication: Can consumer choices and risk-taking be primed by mating motives?" *Journal of Experimental Psychology: General*, 144 (6), e142.

Shimp, Terence A (1994), "Academic appalachia and the discipline of consumer research," *ADVANCES IN CONSUMER RESEARCH, VOL XXI*, ASSOC CONSUMER RESEARCH UNIV MINNESOTA DULUTH, LABOVITZ SCHOOL BUSINESS ….

Simonson, Itamar, Ziv Carmon, Ravi Dhar, Aimee Drolet, and Stephen M Nowlis (2001), "Consumer research: In search of identity," *Annual review of psychology*, 52 (1), 249–75.

Torelli, Carlos J, Ayşegül Özsomer, Sergio W Carvalho, Hean Tat Keh, and Natalia Maehle (2012), "Brand concepts as representations of human values: Do cultural congruity and compatibility between values matter?" *Journal of marketing*, 76 (4), 92–108.

Toubia, Olivier, Jonah Berger, and Jehoshua Eliashberg (2021), "How quantifying the shape of stories predicts their success," *Proceedings of the National Academy of Sciences*, 118 (26), e2011695118.

Toubia, Olivier and Oded Netzer (2017), "Idea generation, creativity, and prototypicality," *Marketing science*, 36 (1), 1–20.

Tucker, Catherine E (2015), "The reach and persuasiveness of viral video ads," *Marketing Science*, 34 (2), 281–96.

Valsesia, Francesca and Kristin Diehl (2022), "Let me show you what i did versus what i have: Sharing experiential versus material purchases alters authenticity and liking of social media users," *Journal of Consumer Research*, 49 (3), 430–49.

Van Heerde, Harald J, Christine Moorman, C Page Moreau, and Robert W Palmatier (2021), "Reality check: Infusing ecological value into academic marketing research," *Journal of Marketing*, SAGE Publications Sage CA: Los Angeles, CA.

Vilalta, Ricardo and Youssef Drissi (2002), "A perspective view and survey of meta-learning," *Artificial intelligence review*, 18, 77–95.

Warren, Caleb, Adam Barsky, and A Peter McGraw (2018), "Humor, comedy, and consumer behavior," *Journal of Consumer Research*, 45 (3), 529–52.

Wedel, Michel and PK Kannan (2016), "Marketing analytics for data-rich environments," *Journal*

of marketing, 80 (6), 97–121.

Weinberger, Marc G, Harlan Spotts, Leland Campbell, and Amy L Parsons (1995), "The use and effect of humor in different advertising media," *Journal of advertising research*, 35 (3), 44–57.

Woltman Elpers, Josephine LCM, Ashesh Mukherjee, and Wayne D Hoyer (2004), "Humor in television advertising: A moment-to-moment analysis," *Journal of Consumer Research*, 31 (3), 592–98.

Wu, Alisa Yinghao and Vicki G Morwitz (2025), "Digital therapy for negative consumption experiences: The impact of emotional and rational reviews on review writers," *Journal of Consumer Research*, 51 (5), 937–58.

Yang, Mochen, Gediminas Adomavicius, Gordon Burtch, and Yuqing Ren (2018), "Mind the gap: Accounting for measurement error and misclassification in variables generated via data mining," *Information Systems Research*, 29 (1), 4–24.

Zantedeschi, Daniel, Eleanor McDonnell Feit, and Eric T Bradlow (2017), "Measuring multichannel advertising response," *Management Science*, 63 (8), 2706–28.

Zhong, Ning and David A Schweidel (2020), "Capturing changes in social media content: A multiple latent changepoint topic model," *Marketing Science*, 39 (4), 827–46.

Zhou, Mi, George H Chen, Pedro Ferreira, and Michael D Smith (2021), "Consumer behavior in the online classroom: Using video analytics and machine learning to understand the consumption of video courseware," *Journal of Marketing Research*, 58 (6), 1079–1100.