CHAPTER III

RESEARCH METHODOLOGY

This chapter presents the systematic research methodology developed to achieve the study objectives. The methodology is structured into two principal phases: **Phase I: Theoretical and Algorithmic Development**, which establishes the mathematical foundations and computational algorithms for Double Generalized Additive Models (DGAMs) enhanced through integration with advanced machine learning techniques, and **Phase II: Computational Implementation and Empirical Validation**, which translates these theoretical developments into executable code and rigorously evaluates model performance using real-world health data spanning critical life-stage transitions.

The research is grounded in a fundamental conceptual framework rooted in Buddhist philosophy and universal human experience: birth, aging, illness, and death. This framework motivates the selection of health datasets representing these four life stages, providing a comprehensive testbed for evaluating DGAM performance across diverse demographic and clinical contexts.

3.1 Phase I: Theoretical and Algorithmic Development

This phase focuses on the mathematical formulation and algorithmic specification of Double Generalized Additive Models (DGAMs) integrated with five machine learning techniques: Smoothing Splines, Explainable Boosting Machines (EBM), Extreme Gradient Boosting (XGBoost), Random Forests (RF), and Support Vector Regression (SVR).

3.1.1 Conceptual Foundation: From Classical GAMs to DGAMs

Limitations of Classical GAMs

In the context of Generalized Linear Models (GLMs) and Generalized Additive Models (GAMs), the response variable Y, conditional on covariates $X_d = x_d$ (where

$X_d = (X_1, \ldots, X_d)^\top$ denotes the d-dimensional covariate vector and x_d is its realized value for a given observation), is typically assumed to follow a distribution from the one-parameter exponential family. Under this framework, the relationship between the mean μ and variance of Y is fully specified:

$$\text{Var}(Y \mid X_d = x_d) = \phi\, b''\big(\theta(x_d)\big), \tag{3.1}$$

where ϕ denotes a constant dispersion parameter and $b(\cdot)$ is the cumulant function of the exponential family distribution as defined in equation (2.1). While mathematically convenient, this assumption can be overly restrictive in practice. Real-world data frequently exhibit several departures from this idealized structure:

- **Overdispersion and Underdispersion:** Count or proportion data often display variance that is larger (overdispersion) or smaller (underdispersion) than that implied by the theoretical distribution.
- **Covariate-Dependent Dispersion:** The degree of variability may itself vary systematically as a function of covariates, a phenomenon not accommodated by constant ϕ.
- **Heteroscedasticity:** Continuous data modeled under normality assumptions may display non-constant variance. For the Gaussian distribution where $b(\theta) = \theta^2/2$, $b''(\theta) = 1$, and $\phi = \sigma^2$, the assumption of constant variance may inadequately reflect the variability observed in real data (Rigby and Stasinopoulos, 2005).

Traditional approaches to addressing these limitations include adopting alternative distributions with more flexible variance structures (e.g., negative binomial or gamma-Poisson models) or applying variance-stabilizing transformations. However, these approaches often require strong distributional assumptions or sacrifice interpretability.

The DGAM Framework

The Double Generalized Additive Model (DGAM) provides a principled extension of classical GAMs by allowing flexible estimation of both mean and dispersion functions. The

response variable Y is assumed to follow a distribution from the exponential dispersion family (equation (2.1)). Unlike classical GAMs that assume constant dispersion, DGAMs model both the location parameter (mean, μ) and scale parameter (dispersion, ϕ) as smooth additive functions of covariates:

$$g(\mu_i) = \eta_{\mu,i} = f_\mu(\mathbf{x}_i) = \beta_0 + \sum_{j=1}^{p} f_{\mu,j}(x_{ij}),$$
$$h(\phi_i) = \eta_{\phi,i} = f_\phi(\mathbf{z}_i) = \gamma_0 + \sum_{j=1}^{q} f_{\phi,j}(z_{ij}),$$

where $g(\cdot)$ and $h(\cdot)$ are link functions, $\mathbf{x}_i = (x_{i1}, x_{i2}, \ldots, x_{ip})^T$ and $\mathbf{z}_i = (z_{i1}, z_{i2}, \ldots, z_{iq})^T$ are covariate vectors (which may be identical or different), and $f_{\mu,j}(\cdot)$ and $f_{\phi,j}(\cdot)$ are smooth functions estimated from the data.

For Gaussian responses with identity link for the mean and log link for the dispersion, this framework takes the form:

$$Y_i \sim \mathcal{N}(\mu_i, \sigma_i^2),$$

with:

$$\mu_i = \beta_0 + \sum_{j=1}^{p} f_{\mu,j}(x_{ij}),$$
$$\log(\sigma_i^2) = \gamma_0 + \sum_{j=1}^{q} f_{\phi,j}(z_{ij}).$$

This dual-structure enables comprehensive characterization of how both central tendency and variability change with predictor variables, making DGAMs particularly powerful for modeling heteroscedastic data where variance depends systematically on covariates.

3.1.2 Mathematical Formulations and Algorithmic Specifications

The theoretical foundations for parameter estimation in DGAMs using each of the five machine learning techniques have been detailed in the preceding sections of this

chapter. These include:

1. **Smoothing Splines:** Penalized likelihood estimation with basis function representations and coupled Penalized Iteratively Reweighted Least Squares (PIRLS) algorithm (Algorithm 1).

2. **Explainable Boosting Machine (EBM):** Alternating boosting on coupled negative log-likelihood gradients with round-robin feature selection (Algorithm 2).

3. **Extreme Gradient Boosting (XGBoost):** Alternating second-order boosting using gradient and Hessian information with regularized tree construction (Algorithm 3).

4. **Random Forest:** Alternating residual forests for mean and dispersion with bootstrap aggregation (Algorithm 4).

5. **Support Vector Regression (SVR):** Alternating optimization with ε-insensitive loss functions and kernel methods (Algorithm 5).

Each technique has been mathematically formulated to accommodate the dual structure of DGAMs, where separate but interdependent models are estimated for the mean μ and dispersion σ^2 functions.

3.1.3 Theoretical Properties and Convergence

The formal mathematical proofs of theoretical properties, including consistency, asymptotic normality, convergence guarantees, and model identifiability conditions for the proposed DGAM estimators, are presented in Chapter 4 alongside empirical results. These theoretical developments establish the statistical foundations supporting the computational implementations developed in Phase II.

3.1.4 Modeling Framework Comparison

Two distinct modeling paradigms are investigated to isolate the impact of dispersion modeling:

GAM with Constant Dispersion

The standard GAM assumes homoscedastic errors:

$$Y_i = \mu_i + \varepsilon_i, \quad \varepsilon_i \sim \mathcal{N}(0, \sigma^2), \tag{3.2}$$

where $\mu_i = \mu(x_i)$ is the mean function for observation i modeled using one of five techniques (Smoothing Splines, EBM, XGBoost, RF, or SVR), and σ^2 is a constant dispersion parameter (identical for all observations) estimated as:

$$\hat{\sigma}^2 = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{\mu}_i)^2 = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{\mu}(x_i))^2,$$

where $\hat{\mu}_i = \hat{\mu}(x_i)$ denotes the fitted mean value for observation i obtained from the trained model. This framework yields 5 distinct GAM configurations, each employing a different technique for mean estimation while maintaining constant variance across all observations.

DGAM with Flexible Dispersion

The Double Generalized Additive Model extends the GAM framework by explicitly modeling heteroscedastic errors:

$$Y_i = \mu_i + \varepsilon_i, \quad \varepsilon_i \sim \mathcal{N}(0, \sigma_i^2),$$

where both $\mu_i = \mu(x_i)$ and $\sigma_i^2 = \sigma^2(x_i)$ are modeled as flexible functions of covariates. Note that the variance σ_i^2 now varies across observations (subscript i), unlike the constant σ^2 in equation (3.2).

Following the GAM structure, the mean and dispersion functions are expressed as

sums of smooth component functions:

$$\mu_i = \mu(x_i) = \beta_0 + \sum_{j=1}^{p} f_{\mu,j}(x_{ij}),$$
$$\log(\sigma_i^2) = f_\phi(x_i) = \gamma_0 + \sum_{j=1}^{q} f_{\phi,j}(z_{ij}),$$

where the dispersion is modeled on the log scale to ensure positivity ($\sigma_i^2 > 0$ for all i), and $f_{\mu,j}(\cdot)$ and $f_{\phi,j}(\cdot)$ are smooth functions estimated from the data.

The DGAM framework allows independent selection of techniques for mean and dispersion modeling, yielding $5 \times 5 = 25$ distinct configurations. This factorial design enables systematic investigation of how different technique combinations affect prediction performance and uncertainty quantification.

3.1.5 Summary of Phase I

Phase I establishes the complete theoretical and algorithmic foundation for DGAMs, including:

- Mathematical formulations for dual mean-dispersion modeling
- Five distinct parameter estimation algorithms
- Theoretical properties and convergence guarantees
- Framework for comparing GAM and DGAM approaches

These theoretical developments provide the foundation for the computational implementation and empirical validation conducted in Phase II.

3.2 Phase II: Computational Implementation and Empirical Validation

This phase translates the theoretical developments from Phase I into executable Python code and conducts comprehensive empirical evaluation using real-world health

datasets. The implementation follows a systematic seven-step procedure encompassing data acquisition, preprocessing, model training, and evaluation.

3.2.1 Step 1: Data Acquisition and Preparation

Dataset Selection and Conceptual Framework

Guided by the universal life-cycle framework (birth, aging, illness, death), four health-related datasets were selected to represent distinct stages of human existence:

1. **Birth Stage Dataset:** Perinatal and neonatal health indicators
   `https://www.kaggle.com/datasets/yaminh/fetal-health-monitoring-dataset`
2. **Aging Stage Dataset:** Age-related physiological and functional measurements
   `https://www.kaggle.com/datasets/jockeroika/life-style-data`
3. **Illness Stage Dataset:** Disease progression and treatment response variables
   `https://www.kaggle.com/datasets/hongseoi/health-checkup-result`
4. **Death Stage Dataset:** Mortality risk factors and end-of-life health metrics
   `https://www.kaggle.com/datasets/lashagoch/life-expectancy-who-updated`

Each dataset contains multiple target variables, yielding a total of 16 distinct prediction tasks (4 datasets $\times$ 4 target variables per dataset). This design ensures comprehensive evaluation across diverse data characteristics and prediction contexts.

Datasets were obtained from Kaggle using the Kaggle API to ensure reproducibility and standardized access protocols. All data acquisition procedures were implemented in Python, with code archived for independent verification.

Data Quality Assessment and Preprocessing

Initial data exploration and quality assessment were performed to identify and address data integrity issues. All preprocessing steps were implemented as reusable Python functions:

Missing Value Treatment:

- **Continuous variables:** Median imputation was employed to preserve distributional properties while minimizing sensitivity to outliers.
- **Categorical variables:** Mode imputation was applied to maintain the most prevalent category representation.

Outlier Detection and Treatment: Extreme values were identified using the Interquartile Range (IQR) method:

$$\text{Outlier if: } x_i < Q_1 - 1.5 \times IQR \text{ or } x_i > Q_3 + 1.5 \times IQR,$$

where Q_1 and Q_3 represent the first and third quartiles, respectively. Identified outliers were retained but flagged for sensitivity analysis.

Categorical Variable Encoding: Categorical variables were transformed using one-hot encoding to generate binary indicator variables suitable for continuous regression modeling, implemented using `scikit-learn`'s `OneHotEncoder`.

3.2.2 Step 2: Data Partitioning Strategy

To assess model stability across different training set sizes, each of the 16 prediction tasks was subjected to three distinct data splitting configurations. All splits employed a fixed random seed (`RANDOM_STATE = 42`) to ensure reproducibility across all computational experiments.

Table 3.1 Data Splitting Configurations

Split Configuration	Training	Validation	Test
Split A	60%	20%	20%
Split B	70%	15%	15%
Split C	80%	10%	10%

This multi-split design, implemented using `scikit-learn`'s `train_test_split` function, enables evaluation of:

- Model performance sensitivity to training set size
- Robustness of conclusions across different data availability scenarios
- Overfitting tendencies as training data increases

The validation set is used for hyperparameter selection (when applicable) and early stopping criteria, while the test set provides unbiased performance estimates for final model comparison.

3.2.3 Step 3: Feature Preprocessing

To ensure numerical stability and facilitate convergence, all continuous features were standardized using z-score normalization, implemented as a custom Python class:

$$z_i = \frac{x_i - \bar{x}_{\text{train}}}{\sigma_{\text{train}}},$$

where $\bar{x}_{\text{train}}$ and σ_{train} are the mean and standard deviation computed from the training set only. Crucially, the same transformation parameters derived from the training data are applied to validation and test sets to prevent information leakage.

Preprocessing parameters are stored as Python objects (using `pickle` or `joblib`) for consistent application during model deployment and to enable proper interpretation of model coefficients and feature importance measures.

3.2.4 Step 4: Model Training Procedures

All model training procedures were implemented as Python classes following object-oriented programming principles, ensuring modularity and reusability.

GAM Training Protocol

For each of the 5 GAM configurations (one per mean modeling technique), training proceeds as follows:

1. Train the mean function $\hat{\mu}(x)$ using the specified technique (Spline, EBM, XGBoost, RF, or SVR) on the training data $\{(x_i, y_i)\}_{i=1}^{n_{\text{train}}}$.
2. Compute residuals:
$$r_i = y_i - \hat{\mu}(x_i), \quad i = 1, \ldots, n_{\text{train}}.$$
3. Estimate constant dispersion via maximum likelihood:
$$\hat{\sigma}^2 = \frac{1}{n_{\text{train}}} \sum_{i=1}^{n_{\text{train}}} r_i^2.$$
This procedure yields a predictive distribution for new observations:
$$\hat{p}(y \mid x) = \mathcal{N}(y; \hat{\mu}(x), \hat{\sigma}^2).$$

DGAM Training Protocol

For each of the 25 DGAM configurations (5 mean techniques $\times$ 5 dispersion techniques), training proceeds through alternating optimization implemented in Python:

1. **Mean Model Training:** Train the mean function $\hat{\mu}(x)$ using the specified mean modeling technique.
2. **Residual Computation:** Calculate squared residuals:
$$r_i^2 = (y_i - \hat{\mu}(x_i))^2, \quad i = 1, \ldots, n_{\text{train}}.$$
3. **Dispersion Model Training:** Train the dispersion function $\log(\hat{\sigma}^2(x))$ on the log-transformed squared residuals:
$$z_i = \log(r_i^2 + \delta),$$
where $\delta = 0.01$ is a small constant ensuring numerical stability.
4. **Iterative Refinement (if applicable):** For techniques supporting iterative updates (EBM, XGBoost), alternate between refining mean and dispersion models until convergence.

The resulting predictive distribution is:
$$\hat{p}(y \mid x) = \mathcal{N}(y; \hat{\mu}(x), \hat{\sigma}^2(x)).$$

Table 3.2 Complete Model Configuration Matrix.

Framework	Mean Technique	Dispersion Modeling Technique					
		Const	Spline	EBM	XGBoost	RF	SVR
GAM	Spline	✓	—	—	—	—	—
	EBM	✓	—	—	—	—	—
	XGBoost	✓	—	—	—	—	—
	RF	✓	—	—	—	—	—
	SVR	✓	—	—	—	—	—
DGAM	Spline	—	✓	✓	✓	✓	✓
	EBM	—	✓	✓	✓	✓	✓
	XGBoost	—	✓	✓	✓	✓	✓
	RF	—	✓	✓	✓	✓	✓
	SVR	—	✓	✓	✓	✓	✓
Total Configurations		5 GAM + 25 DGAM = 30 models per prediction task					

3.2.5 Step 5: Hyperparameter Configuration

To ensure fair comparison and reproducibility, hyperparameters were fixed at empirically stable values rather than extensively tuned for each model configuration. This design decision reflects the study's primary objective: to isolate and quantify the benefit of flexible dispersion modeling within the DGAM framework.

All hyperparameter values were hard-coded in Python configuration files, ensuring consistency across all experiments. The complete hyperparameter settings are summarized in Table 3.3.

Table 3.3 Fixed Hyperparameter Settings for All Experiments.

Model Component	Hyperparameter	Value
Smoothing Splines	Number of knots	10
	Spline degree	3 (cubic)
	Ridge penalty (α)	1.0
	Feature standardization	Enabled (zero mean, unit variance)
EBM	Implementation	`ExplainableBoostingRegressor`
	Random seed	42
	Fallback model	Spline Ridge GAM (if unavailable)
XGBoost	Number of trees	1200
	Maximum tree depth	4
	Learning rate	0.03
	Subsample ratio	0.9
	Column subsample ratio	0.9
	L2 regularization (λ)	1.0
Random Forest (Mean)	Number of trees	800
	Minimum samples per leaf	3
	Bootstrap sampling	Enabled
Random Forest (Disp.)	Number of trees	600
	Minimum samples per leaf	3
	Bootstrap sampling	Enabled
SVR	Kernel	RBF (Radial Basis Function)
	Regularization (C)	10
	Kernel parameter (γ)	`scale`

Rationale for Fixed Hyperparameters

The decision to employ fixed hyperparameters across all configurations is supported by three methodological considerations:

1. **Reproducibility:** Fixed settings ensure exact replication of results across different computational environments and runs, facilitating independent verification.

2. **Fair Comparison:** All model configurations operate under equivalent conditions,

enabling direct attribution of performance differences to the mean-dispersion modeling framework.

3. **Research Focus:** The primary scientific question concerns the value of dispersion modeling in DGAMs, not hyperparameter optimization.

3.2.6 Step 6: Model Evaluation Framework

Each evaluation metric was implemented as a standalone Python function to ensure modularity and clarity in the evaluation pipeline. This structure supports independent testing, easier maintenance, and seamless extension for future metrics.

Point Prediction Metrics

Mean Absolute Error (MAE) MAE provides a straightforward measure of average prediction error:

$$\text{MAE} = \frac{1}{n_{\text{test}}} \sum_{i=1}^{n_{\text{test}}} |y_i - \hat{\mu}(x_i)|.$$

Root Mean Squared Error (RMSE) RMSE measures prediction accuracy with emphasis on larger errors:

$$\text{RMSE} = \sqrt{\frac{1}{n_{\text{test}}} \sum_{i=1}^{n_{\text{test}}} (y_i - \hat{\mu}(x_i))^2}.$$

Coefficient of Determination (R^2) R^2 quantifies the proportion of variance explained by the model:

$$R^2 = 1 - \frac{\sum_{i=1}^{n_{\text{test}}} (y_i - \hat{\mu}(x_i))^2}{\sum_{i=1}^{n_{\text{test}}} (y_i - \bar{y})^2},$$

where $\bar{y} = \frac{1}{n_{\text{test}}} \sum_{i=1}^{n_{\text{test}}} y_i$ is the mean of observed values. Values closer to 1 indicate better model fit.

Likelihood-Based Metrics

For all metrics in this section, let n_{test} denote the number of observations in the test set, y_i the observed value, $\hat{\mu}(x_i)$ the predicted mean, and $\hat{\sigma}(x_i)$ the predicted standard deviation for observation i.

Akaike Information Criterion (AIC) The AIC balances model fit and complexity:

$$\text{AIC} = -2\ln(\hat{L}) + 2k,$$

where $\hat{L}$ is the maximized likelihood and k is the effective number of parameters. Lower AIC values indicate better models.

Negative Log-Likelihood (NLL) The NLL measures the goodness of fit:

$$\text{NLL} = -\sum_{i=1}^{n_{\text{test}}} \log p(y_i \mid x_i; \theta),$$

where $p(y_i \mid x_i; \theta)$ is the probability density function evaluated at y_i given covariates x_i and parameters θ. Lower NLL indicates better predictive performance.

Probabilistic Calibration Metrics

Continuous Ranked Probability Score (CRPS) CRPS evaluates the calibration of probabilistic forecasts:

$$\text{CRPS}(F, y) = \int_{-\infty}^{\infty} \left[F(x) - \mathbf{1}(x \geq y)\right]^2 dx,$$

where $F(\cdot)$ is the predicted cumulative distribution function and $\mathbf{1}(\cdot)$ is the indicator function. The average CRPS over the test set is:

$$\text{CRPS} = \frac{1}{n_{\text{test}}} \sum_{i=1}^{n_{\text{test}}} \text{CRPS}(F_i, y_i).$$

Lower CRPS values indicate better calibrated probabilistic predictions.

Prediction Interval Coverage Probability (PICP_{95}) PICP_{95} measures the empirical coverage of 95% prediction intervals:

$$\text{PICP}_{95} = \frac{1}{n_{\text{test}}} \sum_{i=1}^{n_{\text{test}}} \mathbf{1}(L_i \leq y_i \leq U_i),$$

Model performance is assessed using a comprehensive suite of seven evaluation metrics, all implemented as Python functions, spanning three categories: likelihood-based measures, point prediction accuracy, and probabilistic calibration.

Table 3.4 Comprehensive Evaluation Metrics.

Category	Metric	Notation	Optimal Direction
Point Prediction	Mean Absolute Error	MAE	Minimize (↓)
	Root Mean Squared Error	RMSE	Minimize (↓)
	Coefficient of Determination	R^2	Maximize (↑)
Likelihood-Based	Akaike Information Criterion	AIC	Minimize (↓)
	Negative Log-Likelihood	NLL	Minimize (↓)
Probabilistic	Continuous Ranked Probability Score	CRPS	Minimize (↓)
	Prediction Interval Coverage (95%)	PICP_{95}	≈ 0.95

3.2.7 Step 7: Statistical Analysis and Result Compilation

All evaluation metrics were computed for each of the 1,440 trained models (30 models $\times$ 16 tasks $\times$ 3 splits) and stored in structured data formats (CSV files and Python DataFrames) for subsequent statistical analysis. Python scripts were developed to:

- Aggregate results across splits and tasks
- Compute summary statistics (mean, median, standard deviation)
- Generate comparative visualizations
- Perform statistical significance tests
- Export results for presentation in Chapter 4

3.2.8 Implementation and Computational Infrastructure

All code was implemented in Python 3.9+ using standard scientific computing libraries:

- **Data manipulation:** `pandas`, `numpy`
- **Smoothing Splines:** `scikit-learn`, `scipy.interpolate`
- **EBM:** `interpret` package (InterpretML framework)
- **XGBoost:** `xgboost` library
- **Random Forest:** `scikit-learn.ensemble`
- **SVR:** `scikit-learn.svm`
- **Visualization:** `matplotlib`, `seaborn`
- **Statistical analysis:** `scipy.stats`, `statsmodels`

Complete source code, including data preprocessing pipelines, model training scripts, evaluation procedures, and result compilation scripts, is archived and version-controlled using Git, ensuring full reproducibility and independent verification of results.

Computational experiments were conducted with typical model training times ranging from seconds (for Smoothing Splines and SVR) to minutes (for XGBoost and Random Forests) per configuration, enabling completion of all experiments within reasonable computational budgets.

3.3 Research Framework Summary

To achieve the research objectives, this study adopts a structured two-phase research framework that links theoretical development with computational implementation and empirical evaluation. The framework clarifies how mathematical modeling advances are translated into algorithmic procedures and subsequently validated through systematic experiments.

The overall research design is illustrated in Figures 3.1 and 3.2, showing the progression from theoretical foundations to model implementation and performance assessment.

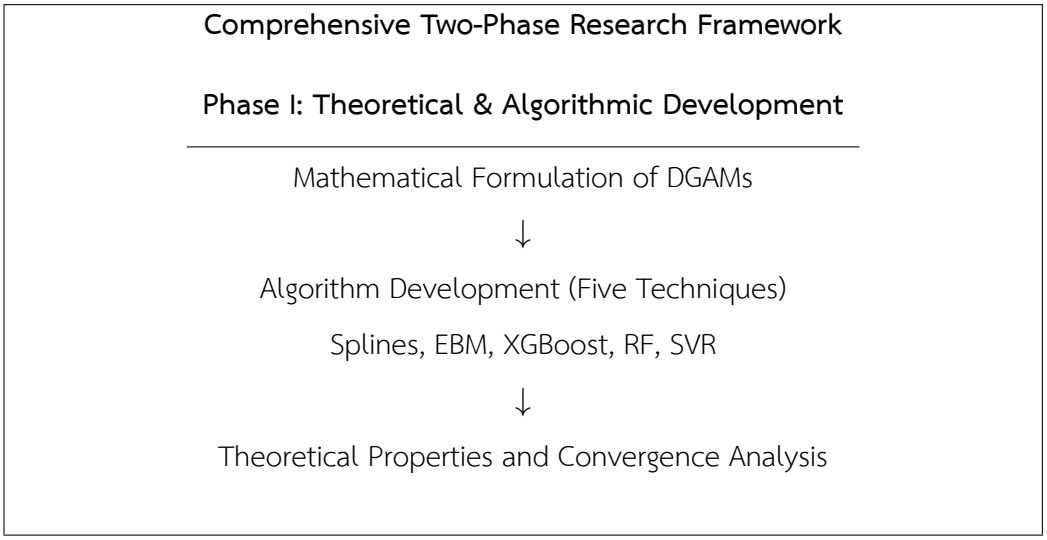

Figure 3.1 Overall Phase I Research Framework: Theoretical and Algorithmic Development.

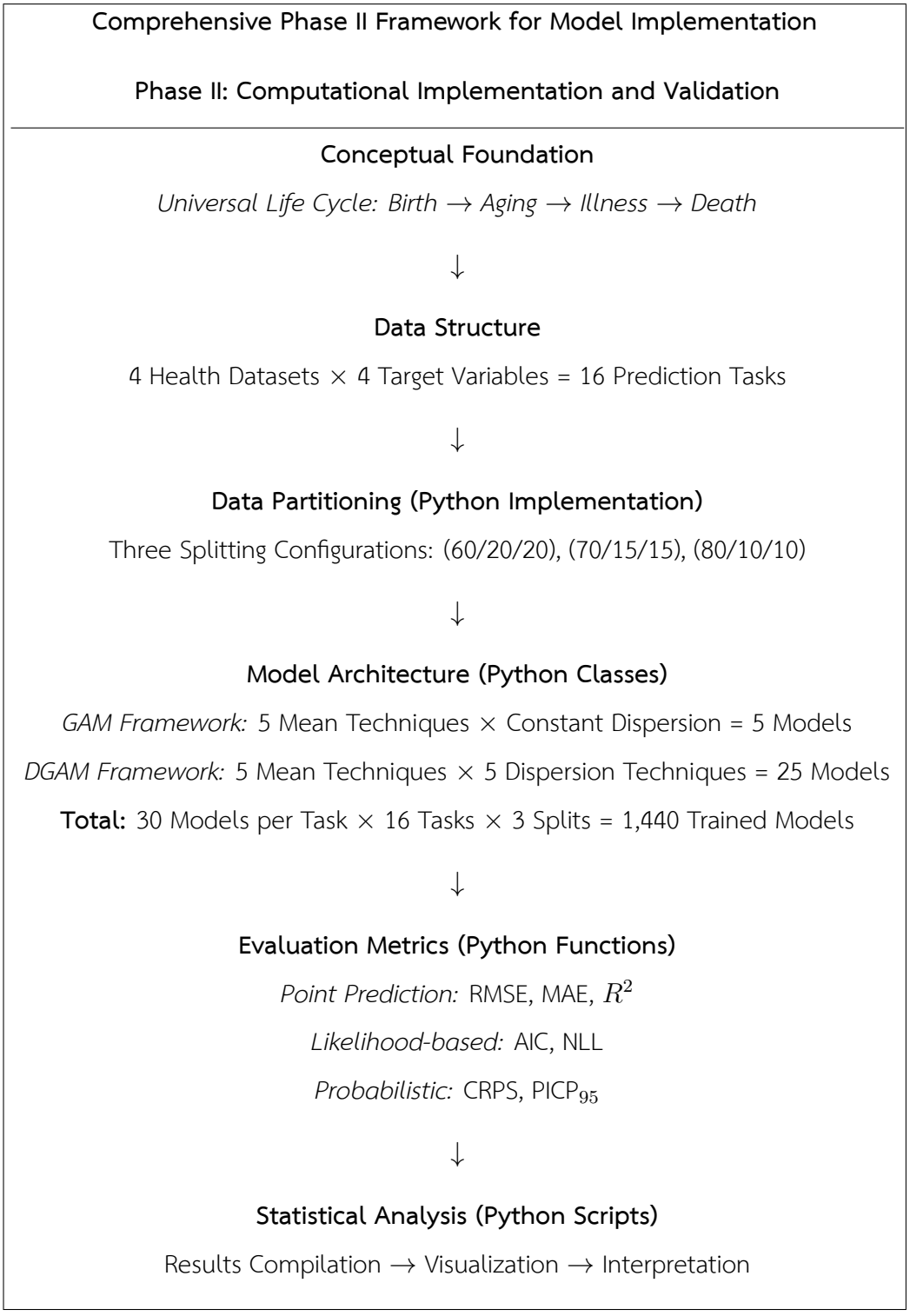

Figure 3.2 Overall Phase II Research Framework: Model Implementation.

This two-phase methodology ensures a rigorous and coherent pathway for the development and evaluation of Double Generalized Additive Models (DGAMs). Phase I establishes the mathematical and algorithmic foundations, including formulation, estimation strategies, and theoretical properties. Phase II operationalizes these advances through structured computational implementation and systematic empirical validation. Together, the phases provide comprehensive evidence regarding the benefits of flexible dispersion modeling in enhancing predictive performance and uncertainty quantification within statistical health analytics.