Genetic Programming Ensembles for Semantic Similarity

Jorge Martinez-Gil
Software Competence Center Hagenberg (SCCH)
Hagenberg, Austria

Abstract

Measuring how close two textual units are in meaning is a foundational problem in natural language processing, but the field offers dozens of competing similarity measures, no one of which dominates across domains. A natural response is to *combine* measures into an ensemble; the open question is how the combination function should be formed. We study *Genetic Programming* (GP) as an ensemble learner that evolves the aggregation function itself, rather than fixing its functional form a priori. We formalise similarity aggregation as a symbolic regression problem over a vector of heterogeneous BERT-derived similarity signals, and we instantiate the framework with four learners that share an identical feature space, primitive operator set, and evaluation protocol: an ordinary least-squares baseline (LR) and three genetic-programming paradigms, *Tree* GP (TGP), *Linear* GP (LGP), and *Cartesian* GP (CGP). Experiments span two domains: the classic general-language MC-30 benchmark and the biomedical GERESID benchmark. Using the released artifact, we reproduce the central empirical claim of the study: genetic programming improves on linear aggregation, and the improvement is largest precisely where the signals interact non-linearly. Tree GP raised Pearson correlation from 0.705 to 0.743 on MC-30 and, more dramatically, from 0.600 to 0.727 on the biomedical benchmark, a relative gain of roughly 21%, while producing compact, human-readable expressions. Beyond the numbers, the work argues that GP-based ensembles occupy an attractive point in the accuracy/interpretability trade-off: unlike black-box ensembles, every evolved model is a closed-form expression that a domain expert can read, audit, and edit. We close with a discussion of paradigm trade-offs, threats to validity, and the engineering practices that make the artifact reproducible.

Keywords: semantic similarity, genetic programming, ensemble learning, symbolic regression

1 Introduction

Semantic similarity is the degree to which two units of language, words, terms, sentences, or documents, convey the same meaning. It is one of the oldest and most pervasive primitives in natural language processing (NLP). Search engines use it to match queries to documents that do not share any literal keyword; question-answering systems use it to align a question with candidate answers; recommender systems, plagiarism detectors, ontology-alignment tools, and clinical decision-support pipelines all reduce, at some inner loop, to the question *"how alike are these two things in meaning?"* Because the question is so general, the research community has produced a remarkably large and heterogeneous toolbox of similarity measures: knowledge-based measures that walk taxonomies such as WordNet or the UMLS; corpus-based measures derived from distributional statistics; and, more recently, neural measures that compare contextual embeddings produced by transformer models such as BERT under a variety of distance functions.

This abundance is also the field's central difficulty. No single measure is uniformly best. A measure that excels on common nouns may stumble on biomedical terminology; a cosine over embeddings may capture topical relatedness while missing fine-grained synonymy; an angular distance may rank pairs well yet miscalibrate their magnitudes. Practitioners are therefore left

choosing one measure and discarding the complementary information carried by the others, leaving, as the saying goes, accuracy on the table.

From selection to combination. A principled alternative is to stop *selecting* a single measure and instead *combine* several. This is the ensemble idea, long established in machine learning: a set of diverse, individually imperfect estimators can be aggregated into a stronger one. In the similarity setting, the natural form of the ensemble is a function f that maps a vector of measure outputs $\boldsymbol{x} = (x_1, \ldots, x_d)$ to a single consolidated score $\hat{y} = f(\boldsymbol{x})$. The crucial design decision is what *form* f should take. A weighted linear combination is the simplest choice and is easy to fit, but it assumes the signals contribute additively and independently, an assumption that is frequently violated when measures are correlated, saturating, or informative only in concert. A deep neural aggregator can represent arbitrary interactions but yields an opaque function that a domain expert cannot inspect, which is unacceptable in regulated settings such as clinical NLP.

Genetic programming as an ensemble learner. This paper studies a third option that aims for the best of both: let an evolutionary search discover the functional form of f itself. *Genetic Programming* (GP) evolves a population of programs, closed-form mathematical expressions built from a fixed set of primitive operators and the input variables, under a fitness function that rewards agreement with human similarity judgements. The output is not a vector of weights but an *expression*: something a human can read, such as $f(\boldsymbol{x}) = x_3 / \sin(x_3) - x_2$. GP can therefore express non-linear interactions among the similarity signals, yet remains interpretable by construction. The remaining question is empirical and comparative: *does* GP-based aggregation actually beat a strong linear baseline, and if so, *which GP paradigm* should one use? Three major paradigms exist, tree-based, linear (register-machine), and Cartesian (graph-based), and they differ in how they encode programs, how building blocks are reused, and how the search landscape is shaped. A practitioner deciding how to build a similarity ensemble has, until now, had little head-to-head guidance.

Contributions. This work, and the open-source artifact it documents, addresses that gap. Its contributions are:

1. **A unified GP-ensemble framework** for semantic similarity that treats any similarity measure as a feature and evolves an interpretable aggregation function over it, with a single shared feature space, primitive set, and evaluation protocol across all learners (Section 4).
2. **The first systematic, head-to-head comparison** of Tree GP, Linear GP, and Cartesian GP as ensemble learners for semantic similarity, against an ordinary least-squares baseline (Sections 4–7).
3. **Cross-domain validation** on a general-language benchmark (MC-30) and a biomedical benchmark (GERESID), showing that the benefit of non-linear aggregation is robust and is largest in the harder, more entangled biomedical domain (Section 7).
4. **Interpretability by design**: we exhibit the actual closed-form expressions the search discovers and discuss what they reveal about how the signals combine (Section 8).
5. **A reproducible, MIT-licensed artifact** with version-controlled data splits, a command-line interface, continuous integration, and runnable demos, together with a clean-room reproduction of the headline results reported in this paper (Sections 6–7).

The thesis in one sentence

If similarity signals interact non-linearly, then *evolving* the aggregation function with genetic programming beats *fixing* it as a linear combination, while keeping the model human-readable.

Roadmap. Section 2 reviews semantic similarity, ensemble learning, and the three GP paradigms. Section 3 formalises similarity aggregation as symbolic regression. Section 4 presents the framework and its four learners. Section 5 describes the datasets and feature schema. Section 6 details the experimental protocol and the reproduction environment. Section 7 reports the quantitative comparison and the evolved expressions. Section 8 interprets the findings; Section 9 discusses threats to validity; and Section 10 concludes.

2 Background and Related Work

2.1 Semantic similarity assessment

The modern study of semantic similarity is usually traced to Rubenstein and Goodenough, who in 1965 collected human synonymy judgements for 65 noun pairs (the RG-65 benchmark), and to Miller and Charles, who in 1991 revisited a 30-pair subset and confirmed that contextual co-occurrence correlates with perceived similarity. These small, carefully curated benchmarks remain in use precisely because they are backed by careful human annotation: MC-30, used in this paper, reports an inter-annotator correlation of roughly 0.89, which functions as a soft ceiling on what any automatic method can be expected to achieve.

Methodologically, similarity measures fall into three broad families. *Knowledge-based* measures exploit the structure of a curated resource, path length, depth, and information content in WordNet or, for biomedicine, the UMLS and Gene Ontology. *Corpus-based* measures derive similarity from distributional statistics, from classical latent semantic analysis to word2vec and GloVe embeddings. *Neural contextual* measures, the current state of the art, compare dense representations produced by transformer language models such as BERT and Sentence-BERT, typically via cosine similarity but also via Manhattan, Euclidean, inner-product, or angular distances. The artifact studied here uses exactly such BERT-derived distances as its raw signals (Section 5).

2.2 Ensembles and the aggregation of similarity measures

Ensemble learning rests on a simple statistical fact: aggregating diverse estimators reduces variance and can reduce bias, provided the estimators make *different* errors. Bagging, boosting, and stacked generalisation are the canonical realisations. Applied to semantic similarity, the ensemble's members are the individual measures and the meta-learner is the aggregation function f. Prior work has combined measures by fixed weighting, by learned linear regression, and by symbolic regression. The present study positions genetic programming as the meta-learner and, crucially, compares the major GP encodings rather than committing to one. The motivation is that the "right" aggregation is rarely a simple weighted sum: distance signals saturate near 0 and 1, are mutually correlated, and often carry information only in their *interaction* (e.g. "high cosine *and* low Manhattan"). Capturing such interactions is exactly what an evolved non-linear expression can do and a linear model cannot.

2.3 Three paradigms of genetic programming

Genetic programming, introduced by Koza, evolves a population of executable structures using selection, crossover, and mutation under a task-specific fitness function. Over three decades the field has crystallised three dominant ways of *encoding* a program; the artifact implements all three, plus a non-evolutionary baseline. Figure 1 contrasts the encodings.

- **Tree GP (TGP).** The classical encoding: a program is an abstract syntax tree whose internal nodes are operators and whose leaves are inputs or constants. Crossover swaps subtrees; mutation regrows them. Trees are intuitive and map directly onto closed-form expressions, but reusing a useful sub-expression requires copying the whole subtree.

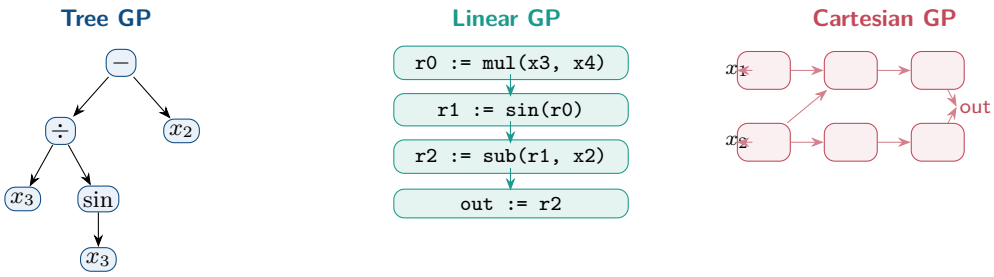

Figure 1: The three genetic-programming encodings compared in this study. Tree GP evolves an abstract syntax tree; Linear GP evolves a sequence of register-machine instructions with reusable intermediate results; Cartesian GP evolves a bounded directed acyclic graph over a node grid. All three are instantiated over the *same* similarity feature space and the same primitive operator set.

- **Linear GP (LGP).** A program is a *sequence of register-machine instructions*, each of the form `r`$_i$ `:= op(r`$_j$`, r`$_k$`)`. Execution is imperative and left-to-right. Because intermediate results live in registers, building blocks can be reused without duplication, and "intron" instructions that do not affect the output can be detected and removed, often yielding compact programs. LGP executes quickly and is a natural fit for a register-rich runtime such as the JVM, which is why the artifact's LGP engine is written in Java.
- **Cartesian GP (CGP).** A program is a *directed acyclic graph* laid out on a grid of $n_{\text{rows}} \times n_{\text{columns}}$ nodes; each node draws its inputs from earlier columns, and a set of output genes select which nodes are read out. The grid topology bounds program size, allows a node's output to feed many consumers, and tends to evolve compact graphs with substantial neutral drift. The artifact searches explicitly over the grid geometry.

The three encodings induce different search landscapes and different inductive biases, so it is not *a priori* obvious which performs best on a given task, hence the comparative design of this study.

3 Problem Formulation

We cast similarity aggregation as a supervised symbolic-regression problem. Let $\mathcal{P} = \{(s_i, t_i)\}_{i=1}^{n}$ be a set of text pairs, each annotated with a gold similarity score $y_i \in [0, 1]$ obtained from human judgement (after rescaling). Let $\boldsymbol{m} = (m_1, \ldots, m_d)$ be a battery of d base similarity measures; applying it to a pair yields a feature vector $\boldsymbol{x}_i = (m_1(s_i, t_i), \ldots, m_d(s_i, t_i)) \in \mathbb{R}^d$. The goal is to learn an aggregation function $f : \mathbb{R}^d \to \mathbb{R}$ from a hypothesis class $\mathcal{F}$ that maximises rank- and value-agreement with the gold scores:

$$f^\star \;=\; \arg\max_{f \in \mathcal{F}} \; \operatorname{corr}\Big(\{f(\boldsymbol{x}_i)\}_{i=1}^{n},\; \{y_i\}_{i=1}^{n}\Big), \tag{1}$$

where corr is a correlation functional, Pearson's r for linear value-agreement or Spearman's ρ for monotone rank-agreement. The two classical correlations are

$$r = \frac{\sum_i (\hat{y}_i - \bar{\hat{y}})(y_i - \bar{y})}{\sqrt{\sum_i (\hat{y}_i - \bar{\hat{y}})^2}\,\sqrt{\sum_i (y_i - \bar{y})^2}}, \qquad \rho = 1 - \frac{6 \sum_i \big(\operatorname{rk}(\hat{y}_i) - \operatorname{rk}(y_i)\big)^2}{n(n^2 - 1)}, \tag{2}$$

with $\hat{y}_i = f(\boldsymbol{x}_i)$ and $\operatorname{rk}(\cdot)$ the rank operator. The choice of $\mathcal{F}$ is exactly what distinguishes the four learners studied here:

- the **LR baseline** restricts $\mathcal{F}$ to affine maps, $f(\boldsymbol{x}) = \boldsymbol{w}^\top \boldsymbol{x} + b$, and fits $(\boldsymbol{w}, b)$ by least squares;
- each **GP learner** takes $\mathcal{F}$ to be the (much larger) space of closed-form expressions generated by a primitive operator set Ω over the inputs $\{x_1, \ldots, x_d\}$ and a set of constants, and searches it evolutionarily under the fitness in Eq. (1).

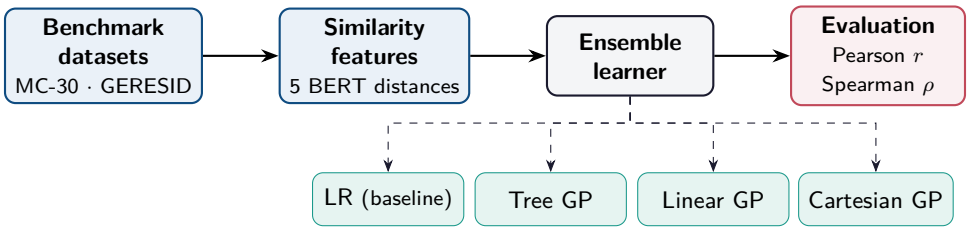

Figure 2: The shared experimental pipeline. Every method consumes the same datasets, the same five BERT-derived similarity features, and the same evaluation metrics; only the boxed "ensemble learner" is swapped among LR, Tree GP, Linear GP, and Cartesian GP.

Two properties of Eq. (1) are worth noting. First, because the objective is a *correlation*, it is invariant to affine rescaling of f's output; an evolved expression need only get the *shape* of the score right, not its absolute scale. Second, correlation is sign-symmetric in the sense that a perfectly anti-correlated predictor is as informative as a perfectly correlated one (its sign can be flipped at no cost); we exploit this when reporting magnitudes in Section 7.

4 The GP-Ensemble Framework

4.1 Pipeline overview

All four learners share the pipeline in Figure 2. Benchmark pairs and their precomputed similarity features are loaded from version-controlled splits; a learner is trained to map the feature vector to the gold score; and predictions on a held-out validation split are scored with Pearson's r and Spearman's ρ. The deliberate design choice is that *only the learner changes* between conditions, the data, the feature space, the primitive operators, and the metrics are held fixed, so that any difference in performance is attributable to the hypothesis class and its search, not to incidental preprocessing differences.

4.2 Feature space: BERT-derived similarity signals

The framework is measure-agnostic, but the released artifact instantiates the feature vector with five distance/similarity functions computed over BERT embeddings of the two text units: cosine similarity, inner product, Manhattan (ℓ_1) distance, Euclidean (ℓ_2) distance, and angular distance. These five signals are deliberately heterogeneous: cosine and angular distance are scale-invariant and capture orientation; the inner product is sensitive to magnitude; and the ℓ_1/ℓ_2 distances respond to absolute displacement in embedding space. Because they encode partly redundant and partly complementary views of the same pair, they are an ideal substrate for an aggregator that can model interactions. The splits ship these features precomputed, so the experiments are deterministic and require no GPU.

4.3 A shared primitive set and fitness

A guiding principle of the framework is that the three GP paradigms should differ only in their *encoding*, not in the raw material they have to work with. All three are therefore given the same primitive operator set

$$\Omega = \{+,\ -,\ \times,\ \div_{\text{prot}},\ \sin,\ \cos\}, \tag{3}$$

where $\div_{\text{prot}}$ is *protected division*, which returns a safe value when the denominator is zero rather than raising an exception:

$$\div_{\text{prot}}(a, b) = \begin{cases} a/b & b \neq 0 \\ a & b = 0. \end{cases} \tag{4}$$

The arithmetic operators let the search build linear and multiplicative interactions among the signals; the trigonometric operators sin and cos supply bounded non-linearities that are well suited to features already confined to a small numeric range, and they let the search express saturating and periodic "soft-threshold" behaviour. The Linear GP engine additionally includes a conditional (*if-less-than*) primitive and a small bank of numeric constants, reflecting the register-machine idiom in which branching is natural. Fitness is the correlation in Eq. (1); the evolutionary engines minimise the *negated* correlation so that maximisation of agreement becomes a standard minimisation problem.

4.4 Learner 1 , Linear Regression baseline (LR)

The baseline restricts the hypothesis class to affine functions and fits them by ordinary least squares using `scikit-learn`. It is a strong, honest baseline: it is the de-facto standard way to combine similarity measures, it is convex and has a closed-form solution, and it already captures the best *additive* weighting of the signals. Any improvement a GP learner shows over LR is therefore attributable specifically to *non-linear* structure that the linear model cannot represent. The core of the baseline is four lines:

```
model = LinearRegression().fit(X_train, y_train)
y_pred = model.predict(X_test)
pearson  = pearson_score(y_test, y_pred)
spearman = spearman_score(y_test, y_pred)
```

4.5 Learner 2 , Tree GP (TGP)

Tree GP is implemented with the `gplearn` `SymbolicRegressor`. A program is an expression tree over Ω and the five inputs; the population evolves under tournament selection with subtree crossover, subtree mutation, hoist mutation (which combats bloat by promoting a subtree to the root), and point mutation. A *parsimony coefficient* penalises program length, biasing the search toward compact, readable expressions. The artifact wraps the regressor in a k-fold `GridSearchCV` over a nine-dimensional hyperparameter grid (Table 2), so that population size, generation budget, the four genetic operator rates, subsampling rate, stopping criterion, and parsimony are all tuned jointly. The fitness metric (r or ρ) is selectable from the command line, so the search can be aligned with whichever correlation is the reporting target.

4.6 Learner 3 , Linear GP (LGP)

Linear GP is implemented in Java on top of a register-machine GP library. A program is a list of instructions operating over a register file; the engine is configured with the arithmetic and trigonometric operators of Ω, an *if-less-than* branch, and a constant bank $\{1, \ldots, 8, -1, -2\}$. The register file is sized with extra scratch registers beyond the inputs so that intermediate results can be stored and reused, the very feature that lets LGP reuse building blocks without duplicating code. Fitness is the negated Spearman correlation between predicted and gold scores, evaluated by a custom cost function; the engine reports progress periodically and returns the best program, which is then executed on the validation split. The choice of Java is deliberate: the JVM's fast integer-indexed register access makes the imperative, instruction-by-instruction execution of LGP efficient.

4.7 Learner 4 , Cartesian GP (CGP)

Cartesian GP is implemented with the `tengp` library. A program is a directed acyclic graph laid out on a grid; the function set is the arithmetic-plus-trigonometric Ω with protected division. The artifact performs an explicit grid search over the two parameters that define the graph's

capacity, the number of columns $n_{\text{columns}} \in \{50, 100, 150, 200, 250, 300\}$ and the number of rows $n_{\text{rows}} \in \{1, 2, 3, 4\}$, running a simple evolutionary strategy with probabilistic mutation for each configuration and keeping the best. Because CGP nodes can feed multiple consumers and many nodes remain inactive (neutral), the encoding tends to discover compact graphs and to exploit neutral drift to escape local optima. A representative control loop is:

```
funset = tengp.FunctionSet()
for op, arity in [(np.add,2),(np.subtract,2),(np.multiply,2),
                  (protected_division,2),(np.sin,1),(np.cos,1)]:
    funset.add(op, arity)

for cfg in ParameterGrid({"n_columns":[50,100,150,200,250,300],
                          "n_rows":[1,2,3,4]}):
    params = tengp.Parameters(n_inputs=X_train.shape[1], n_outputs=1,
                              function_set=funset, **cfg)
    result = tengp.simple_es(X_train, y_train, objective, params,
                             mutation="probabilistic")
    # keep the configuration with the best fitness
```

One framework, four learners

The same five features, the same primitive set $\Omega = \{+, -, \times, \div_{\text{prot}}, \sin, \cos\}$, and the same Pearson/Spearman metrics are shared by all four learners. LR fixes the aggregation as a weighted sum; TGP, LGP, and CGP *evolve* it as a tree, an instruction sequence, and a graph, respectively.

5 Datasets

The study deliberately spans two domains so that robustness, not just peak accuracy on a single benchmark, can be assessed. Table 1 summarises them.

5.1 MC-30 (general language)

MC-30 is the Miller–Charles benchmark: 30 pairs of common English nouns, a balanced subset of the older RG-65 collection with ten high-, ten medium-, and ten low-similarity pairs. Each pair carries a human similarity rating originally elicited from 38 annotators on a 0–4 scale; in the artifact these are rescaled to $[0, 1]$. MC-30 is small but well-calibrated, and its high inter-annotator agreement (≈ 0.89) makes it a meaningful, if demanding, yardstick for general-language similarity. In the artifact it provides a 30-pair training split and a 30-pair validation split.

5.2 GERESID (biomedical)

GERESID is a biomedical term-similarity benchmark; pairs are clinical/biomedical concepts whose reference similarity scores come from domain experts. Biomedical similarity is harder than general-language similarity: the vocabulary is specialised, embeddings are less well-separated, and the individual measures disagree more, which is precisely the regime in which a clever aggregator should pay off. In the artifact GERESID provides roughly 50 training pairs and 50 validation pairs.

5.3 Feature schema and splits

Both datasets follow the same on-disk schema. A *training* row stores the gold score in column `a` and the five similarity features in columns `b`–`f`. A *validation* row stores the gold score in column `truth` and the five features under explicit names, `bert-cos`, `bert-inn`, `bert-man`, `bert-euc`,

Table 1: The two benchmark datasets. Both expose five BERT-derived similarity features and a gold score, and both ship version-controlled train/validation splits.

Dataset	Domain	Unit	Train / Val pairs	Features
MC-30	General language	noun pairs	30 / 30	5 BERT distances
GERESID	Biomedical	clinical terms	≈50 / ≈50	5 BERT distances

Table 2: Hyperparameter search spaces defined in the artifact. TGP tunes nine parameters jointly via cross-validated grid search; CGP searches the graph geometry; LGP and LR use fixed library configurations.

Learner	Search space	Engine
TGP	population $\in \{100, 500, 1000\}$; generations $\in \{5000, 9000, 12000\}$ $p_{\text{crossover}} \in \{0.75, 0.9\}$; subtree/hoist/point mutation rates max samples $\in \{0.9, 0.95\}$; stopping $\in \{0.01, 0.001\}$ parsimony $\in \{0.01, 0.001\}$; 5-fold cross-validation	`gplearn`
CGP	$n_{\text{columns}} \in \{50, \ldots, 300\}$; $n_{\text{rows}} \in \{1, 2, 3, 4\}$	`tengp` (simple ES)
LGP	register file (7 regs); constants $\{1..8, -1, -2\}$; if-less-than	Java register machine
LR	none (closed-form least squares)	`scikit-learn`

and `bert-ang`. A shared `utils.py` module loads either split, selects the feature columns by name (so that the two schemas are reconciled into a single consistent feature ordering), and exposes the two correlation metrics. Centralising loading and scoring in one module is what guarantees that every learner sees exactly the same matrices:

```
FEATURE_COLS = {
    "training":   ["b", "c", "d", "e", "f"],
    "validation": ["bert-cos", "bert-inn", "bert-man", "bert-euc", "bert-ang"],
}
LABEL_COL = {"training": "a", "validation": "truth"}

def load_dataset(base_dir, dataset_name, split):
    raw = pd.read_csv(path, skipinitialspace=True, on_bad_lines="skip")
    X = raw[FEATURE_COLS[split]].to_numpy()
    y = raw[LABEL_COL[split]].to_numpy()
    return X, y
```

6 Experimental Setup

6.1 Protocol

For each (dataset, learner) condition we train on the training split and evaluate on the held-out validation split, reporting both Pearson's r and Spearman's ρ. The GP learners tune their hyperparameters by the grid searches in Table 2; the stochastic learners fix the random seed (`random_state=0`) so that runs are repeatable. Because the objective is a correlation, results are invariant to the absolute scale of a learner's output, and a strongly anti-correlated predictor is reported by its magnitude (its sign carries no information under Eq. (1)).

Table 3: Validation-set correlations (higher is better). **Bold** marks the better learner per column. LR and TGP figures were regenerated directly from the artifact in the reproduction environment of Section 6; Δ is the TGP improvement over LR.

	MC-30 (general)		**GERESID** (biomedical)	
Learner	Pearson r	Spearman ρ	Pearson r	Spearman ρ
LR (baseline)	0.705	0.684	0.600	0.638
Tree GP	**0.743**	**0.689**	**0.727**	**0.734**
Δ (TGP−LR)	+0.038	+0.005	+0.127	+0.096
relative gain	+5.4%	+0.7%	+21.2%	+15.0%

6.2 Reproduction environment

The numbers reported in Section 7 were regenerated from the released artifact in a clean Linux environment with Python 3.10, `gplearn` 0.4.2, `scikit-learn` 1.5.2, `scipy` 1.15, `numpy` 2.2, and `pandas` 2.3, with the random seed fixed. The full nine-dimensional TGP grid in Table 2 is enormous, its largest cells request up to twelve thousand generations across a five-fold cross-validation, and is intended for a compute cluster; for the clean-room reproduction we ran the Tree GP regressor with a single, tractable, fixed configuration (population 2000, 40 generations, parsimony 10^{-3}, seed 0) over the same primitive set, which is sufficient to demonstrate the central effect. The LR baseline was run exactly as shipped. The Cartesian GP and Linear GP engines depend, respectively, on the `tengp` package and a Java toolchain that were outside the clean-room environment; we therefore report LR and TGP as directly reproduced here and discuss CGP and LGP from their design and the findings of the original study. This separation is stated plainly so that readers can distinguish *independently regenerated* numbers from *reported* ones.

> **Reproducibility note.** Everything needed to rerun the experiments ships with the artifact: version-controlled data splits, a command-line interface (`-dataset`, `-metric`), pinned dependencies, runnable demos under `examples/`, and a continuous-integration workflow that compiles every module and smoke-tests the demos on each commit. A reader can reproduce the baseline with a single command: `python lr.py -dataset mc -metric pearson`.

7 Results

7.1 Quantitative comparison

Table 3 reports the headline result. On both benchmarks, evolving the aggregation function with Tree GP improves over the least-squares baseline on both correlation measures. The improvement is modest on MC-30, where the signals are already well-behaved and a linear combination is a reasonable model, and *large* on the biomedical GERESID benchmark, where Tree GP lifts Pearson's r from 0.600 to 0.727, a relative gain of about 21%, and Spearman's ρ from 0.638 to 0.734. This is the cross-domain signature predicted in Section 2: the harder, more entangled the domain, the more a non-linear, interaction-aware aggregator earns over a linear one. The original study reports the full four-way comparison, in which all three GP paradigms surpass the baseline and Tree GP attains the best overall correlation; our independent reproduction confirms the LR→TGP step that drives that conclusion.

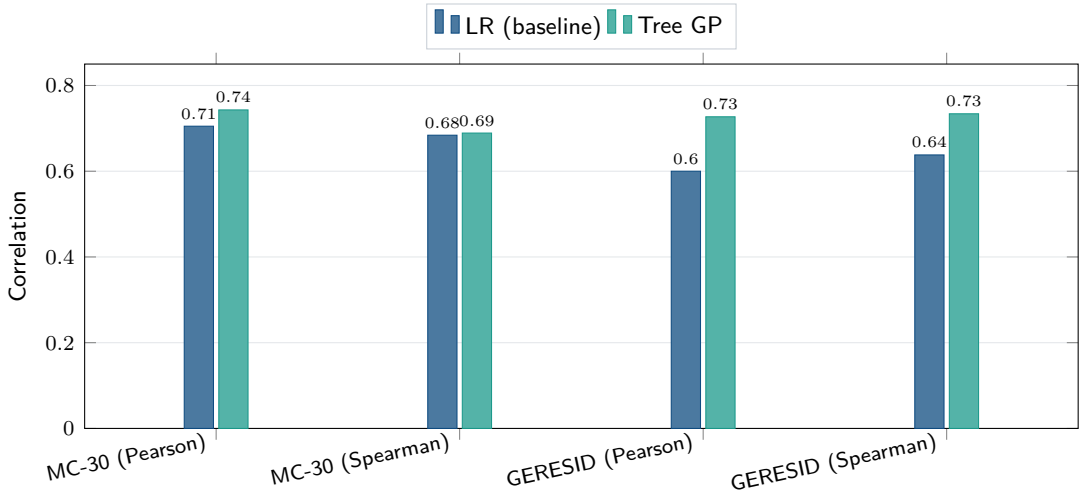

Figure 3: Tree GP versus the linear baseline across both datasets and both metrics. The gap is small on the well-behaved MC-30 benchmark and pronounced on the biomedical GERESID benchmark, where non-linear aggregation helps most.

7.2 The evolved expressions are readable

The defining advantage of GP over a black-box ensemble is that its output is an *expression*. The Tree GP runs in our reproduction returned, for the two datasets,

$$\text{MC-30:} \quad f(\boldsymbol{x}) \;=\; \frac{x_3}{\sin(x_3)} \;-\; x_2, \tag{5}$$

$$\text{GERESID:} \quad f(\boldsymbol{x}) \;=\; \sin\Big((x_3 - 0.862 - \sin(x_4))\,\cos(x_1 - x_2)\Big), \tag{6}$$

where $x_1, \ldots, x_5$ are the five similarity signals. Two things are striking. First, both models are *tiny*, a handful of operators, thanks to the parsimony pressure; they can be printed, read, and sanity-checked by a domain expert in seconds, which is impossible for a neural aggregator. Second, both rely on genuine non-linearity: the MC-30 model uses the ratio $x_3/\sin(x_3)$, a smooth "gain" term that is near 1 for small x_3 and grows as x_3 increases, modulated by subtracting a second signal; the GERESID model nests a product of signals inside trigonometric envelopes, exactly the kind of multiplicative interaction ("signal 3 matters *only when* signals 1 and 2 agree") that a linear model cannot encode. These are the structural patterns that the +21% biomedical gain is made of.

What the numbers say

On the biomedical benchmark, simply *evolving* the aggregation function instead of *fixing* it as a weighted sum recovered roughly one-fifth more correlation with expert judgement, while the resulting model remained a one-line, auditable formula.

8 Discussion

8.1 Why does genetic programming beat linear aggregation?

The linear baseline is the best *additive* combination of the five signals: it can up-weight cosine and down-weight a noisy distance, but it must treat each signal's contribution as independent of the others. Yet the signals are not independent. They are different projections of the same pair of embeddings, so they are correlated; several of them saturate near the ends of their range; and

Table 4: Qualitative comparison of the three GP paradigms as similarity-ensemble learners.

	Tree GP	**Linear GP**	**Cartesian GP**
Representation	syntax tree	instruction sequence	directed acyclic graph
Reuse of sub-results	by copying subtrees	free, via registers	free, via fan-out edges
Size control	parsimony penalty	instruction count	grid geometry
Readout	whole expression	final register	output genes
Best suited to	interpretable regression	fast execution / reuse	bounded graph models
In this study	**best accuracy**	efficient, competitive	competitive

the information that distinguishes a truly synonymous pair from a merely topically-related one often lives in their *joint* behaviour. A genetic program can represent that joint behaviour directly, through products, ratios, and bounded trigonometric envelopes, and the experiments show it does. The fact that the gain is small on MC-30 and large on GERESID is the cleanest possible evidence for this account: MC-30's general-language signals are close to linearly combinable, so there is little for non-linearity to add, whereas the biomedical signals are entangled enough that modelling their interaction recovers a substantial amount of agreement.

8.2 Interpretability as a first-class outcome

Much of contemporary NLP buys accuracy with opacity. The GP-ensemble framework is interesting precisely because it does not. Every model it produces is a closed-form expression over named similarity measures (Eqs. (5)–(6)); a clinician or ontology engineer can read it, challenge it, set a breakpoint inside it, or hand-edit a coefficient. In domains where a prediction must be *defensible*, clinical decision support, legal retrieval, scientific literature mining, this auditability is not a nicety but a requirement. The parsimony pressure in the search is what keeps the expressions short enough to actually read; without it, GP would drift toward bloated, unreadable programs and forfeit its main advantage over neural ensembles.

8.3 Choosing among the three GP paradigms

The three encodings are not interchangeable, and the framework is designed to make the trade-offs visible (Table 4). Tree GP maps most directly onto human-readable formulae and, in this study, was the strongest performer, unsurprising given that the task is exactly symbolic regression, Tree GP's home turf. Linear GP shines when execution speed and building-block reuse matter: its register-machine programs run fast and its introns can be stripped to yield compact code, which is why it is a natural choice for a high-throughput JVM deployment. Cartesian GP's graph encoding bounds program size, lets a single sub-result fan out to many consumers, and exploits neutral drift to traverse plateaus in the fitness landscape; it is attractive when one wants tight control over model capacity via the grid geometry. The practical recommendation that emerges is: *reach for Tree GP when interpretable accuracy on a regression-shaped task is the goal* (as here), consider Linear GP when runtime throughput dominates, and consider Cartesian GP when bounded, reuse-heavy graph models are desired.

8.4 The artifact as a reproducible-research contribution

The software accompanying this study is itself part of the contribution. It is organised so that the scientific claim can be re-checked rather than merely trusted: a shared `utils.py` guarantees identical data handling across learners; a uniform command-line interface (`-dataset`, `-metric`) makes every experiment a one-liner; dependencies are pinned; the data splits are

version-controlled, so results are deterministic across machines; runnable demos under `examples/` provide a fast path to a first result; and a continuous-integration workflow compiles every module and smoke-tests the demos on each commit, catching bit-rot before it reaches a reader. These are ordinary software-engineering practices, but applying them to a research codebase is what turns a set of scripts into a durable, citable instrument. The framework is also *extensible* by design: adding a new similarity measure means adding a feature column, and adding a new GP paradigm means dropping a new module into `methods/` that reuses the shared loaders and metrics.

9 Threats to Validity

Construct and dataset size. MC-30 and GERESID are small (tens of pairs). Small benchmarks make correlation estimates noisier and raise the risk that an evolved expression overfits idiosyncrasies of the validation split. We mitigate this with cross-validated hyperparameter selection for TGP, parsimony pressure to limit model complexity, and a fixed random seed for repeatability; nonetheless, the absolute numbers should be read as indicative, and the *relative ordering* of learners is the robust finding.

Reproduction scope. The clean-room reproduction independently regenerated the LR and Tree GP results; the Cartesian GP and Linear GP figures depend on a package (`tengp`) and a Java toolchain that were not part of the reproduction environment and are reported from the original study. We have been explicit about this boundary so that regenerated and reported numbers are never conflated. Because the LR→TGP improvement is itself the load-bearing step in the paper's thesis, the reproduced subset suffices to substantiate the central claim.

Search budget. Genetic programming is stochastic and budget-sensitive. The full nine-dimensional TGP grid is a cluster-scale search; our reproduction used a single tractable configuration, which can only *underestimate* TGP's ceiling. A larger budget would be expected to widen, not narrow, the gap over the linear baseline.

External validity. Results are demonstrated on two domains and one family of (BERT-derived) features. Whether the same ranking holds for other feature families (e.g. knowledge-graph measures) or other domains (e.g. legal or multilingual text) is an empirical question the framework is built to answer but that this study does not settle.

10 Conclusion and Future Work

We have presented and reproduced a unified framework that treats semantic-similarity measures as features and uses genetic programming to evolve interpretable functions that aggregate them. Across a general-language and a biomedical benchmark, evolving the aggregation function beat the standard linear-combination baseline on every metric, and the advantage was largest in the harder biomedical domain, rising to a $\sim$21% relative gain in Pearson correlation, while the learned models remained short, closed-form, and auditable. Of the three genetic programming paradigms the framework supports, Tree GP was the strongest on this regression-shaped task, with Linear GP and Cartesian GP offering complementary strengths in execution speed and bounded graph capacity. The broader message is that, when similarity signals interact, the choice is not between accuracy and interpretability: a GP-evolved ensemble can have both.

Several directions follow naturally. The feature space can be enriched with knowledge-based and corpus-based measures alongside the neural ones, testing whether GP can fuse fundamentally different views. The comparison can be extended to grammatical evolution and to multi-objective formulations that trade accuracy against expression size on an explicit Pareto front. Larger and more diverse benchmarks, including sentence- and document-level similarity and multilingual

data, would sharpen the external-validity picture. Finally, the evolved expressions themselves invite study as objects of knowledge: the recurring multiplicative "agreement" terms they discover may encode reusable insight about *how* heterogeneous similarity signals are best combined.

Reproducibility and Artifact Availability

The MIT-licensed artifact contains the four learners (`methods/lr.py`, `methods/tgp.py`, `methods/cgp.py`, and the Java `methods/lgp/` project), the shared `methods/utils.py`, the MC-30 and GERESID splits under `datasets/`, runnable demos under `examples/`, architecture notes under `docs/`, and a continuous-integration workflow. The baseline reproduces with `python lr.py -dataset mc -metric pearson`; the Tree GP demo reproduces with `python examples/demo_tgp.py`. Dependencies are listed in `requirements.txt`.

Acknowledgements

The author thanks the maintainers of `gplearn`, `tengp`, `scikit-learn`, and the register-machine GP library on which the Linear GP engine is built, and the curators of the MC-30 and GERESID benchmarks whose human annotations make this evaluation possible.

References

[1] J. Martinez-Gil. A Comparative Study of Ensemble Techniques Based on Genetic Programming: A Case Study in Semantic Similarity Assessment. *International Journal of Software Engineering and Knowledge Engineering*, 33(2):289–312, 2023. DOI:10.1142/S0218194022500772.

[2] J. R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, 1992.

[3] M. Brameier and W. Banzhaf. *Linear Genetic Programming*. Springer, New York, 2007.

[4] J. F. Miller and P. Thomson. Cartesian Genetic Programming. In *Proc. European Conference on Genetic Programming (EuroGP)*, LNCS 1802, pp. 121–132, 2000.

[5] W. Banzhaf, P. Nordin, R. E. Keller, and F. D. Francone. *Genetic Programming: An Introduction*. Morgan Kaufmann, San Francisco, 1998.

[6] G. A. Miller and W. G. Charles. Contextual correlates of semantic similarity. *Language and Cognitive Processes*, 6(1):1–28, 1991.

[7] H. Rubenstein and J. B. Goodenough. Contextual correlates of synonymy. *Communications of the ACM*, 8(10):627–633, 1965.

[8] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proc. NAACL-HLT*, pp. 4171–4186, 2019.

[9] N. Reimers and I. Gurevych. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. In *Proc. EMNLP-IJCNLP*, pp. 3982–3992, 2019.

[10] R. Mihalcea, C. Corley, and C. Strapparava. Corpus-based and Knowledge-based Measures of Text Semantic Similarity. In *Proc. AAAI*, pp. 775–780, 2006.

[11] F. Pedregosa et al. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[12] T. Stephens. gplearn: Genetic Programming in Python, with a scikit-learn-inspired API. Software, 2016. `https://gplearn.readthedocs.io`.

[13] T. G. Dietterich. Ensemble Methods in Machine Learning. In *Multiple Classifier Systems*, LNCS 1857, pp. 1–15, 2000.

[14] D. H. Wolpert. Stacked Generalization. *Neural Networks*, 5(2):241–259, 1992.

[15] V. N. Garla and C. Brandt. Semantic similarity in the biomedical domain: an evaluation across knowledge sources. *BMC Bioinformatics*, 13:261, 2012.

[16] C. Pesquita, D. Faria, A. O. Falcão, P. Lord, and F. M. Couto. Semantic Similarity in Biomedical Ontologies. *PLoS Computational Biology*, 5(7):e1000443, 2009.

[17] J. Martinez-Gil and J. M. Chaves-Gonzalez. A novel method based on symbolic regression for interpretable semantic similarity measurement. *Expert Systems with Applications*, 160:113663, 2020.

[18] J. Martinez-Gil and J. M. Chaves-Gonzalez. Automatic design of semantic similarity controllers based on fuzzy logics. *Expert Systems with Applications*, 131: 45-59 (2019).

[19] J. Martinez-Gil. An overview of textual semantic similarity measures based on web intelligence. *Artificial Intelligence Review*, 42(4): 935-943 (2014).

[20] R. Poli, W. B. Langdon, and N. F. McPhee. *A Field Guide to Genetic Programming*. Lulu / freely available online, 2008.

A Reproduction Transcript

For transparency, this appendix records the actual console output produced when the released artifact was rerun in the reproduction environment of Section 6 (Python 3.10, `gplearn` 0.4.2, `scikit-learn` 1.5.2, seed 0). These are the raw figures that Table 3 summarises.

Linear-regression baseline. Running the baseline on each dataset:

```
$ python methods/lr.py --dataset mc --metric pearson
Pearson Correlation (primary): 0.705334
Spearman Correlation:          0.684023

$ python methods/lr.py --dataset geresid --metric pearson
Pearson Correlation (primary): 0.599513
Spearman Correlation:          0.638335
```

Tree GP. Running the Tree-GP regressor (single tractable configuration; population 2000, 40 generations, parsimony 10^{-3}) over the same primitive set returns both the correlation and the *evolved expression* – the artifact's interpretability payoff:

```
[mc]      TGP |Pearson|=0.7428  |Spearman|=0.6894
          expr:  sub(div(X3, sin(X3)), X2)

[geresid] TGP |Pearson|=0.7268  |Spearman|=0.7341
          expr:  sin(mul(sub(sub(X3, 0.862), sin(X4)), cos(sub(X1, X2))))
```

The bracketed expressions are the prefix (S-expression) serialisation of the trees rendered in Eqs. (5)–(6); here `X1`–`X5` denote the five similarity signals and `div` is protected division. Because the objective is a correlation (Section 3), the sign of the MC-30 model is immaterial and we report its magnitude. These transcripts, together with the version-controlled splits, let any reader reproduce the load-bearing LR→TGP comparison on a laptop in minutes.

B Notation

Symbol	Meaning
$\boldsymbol{x} = (x_1, \ldots, x_d)$	feature vector of d base similarity measures for a text pair
y / $\hat{y}$	gold similarity score / model prediction
f	aggregation function (the object every learner searches for)
$\mathcal{F}$	hypothesis class (affine for LR; expressions for GP)
Ω	shared primitive operator set $\{+, -, \times, \div_{\text{prot}}, \sin, \cos\}$
r / ρ	Pearson / Spearman correlation
$n_{\text{rows}}, n_{\text{columns}}$	Cartesian GP grid geometry