From Hardcoded Heuristics to Graph-Theoretical Constructs: A Principled Reformulation of the Copycat Architecture

Alex Linhares

January 28, 2026

Abstract

The Copycat architecture, developed by Mitchell and Hofstadter as a computational model of analogy-making, relies on numerous hardcoded constants and empirically-tuned formulas to regulate its behavior. While these parameters enable the system to exhibit fluid, human-like performance on letter-string analogy problems, they also introduce brittleness, lack theoretical justification, and limit the system's adaptability to new domains. This paper proposes a principled reformulation of Copycat's core mechanisms using graph-theoretical constructs. We demonstrate that many of the system's hardcoded constants—including bond strength factors, salience weights, and activation thresholds—can be replaced with well-studied graph metrics such as betweenness centrality, clustering coefficients, and resistance distance. This reformulation provides three key advantages: theoretical grounding in established mathematical frameworks, automatic adaptation to problem structure without manual tuning, and increased interpretability of the system's behavior. We present concrete proposals for substituting specific constants with graph metrics, analyze the computational implications, and discuss how this approach bridges classical symbolic AI with modern graph-based machine learning.

1 Introduction

Analogy-making stands as one of the most fundamental cognitive abilities, enabling humans to transfer knowledge across domains, recognize patterns in novel situations, and generate creative insights. Hofstadter and Mitchell's Copycat system [6, 8] represents a landmark achievement in modeling this capacity computationally. Given a simple analogy problem such as "if abc changes to abd, what does ppqqrr change to?," Copycat constructs representations, explores alternatives, and produces answers that exhibit remarkable

similarity to human response distributions. The system's architecture combines a permanent semantic network (the Slipnet) with a dynamic working memory (the Workspace), coordinated through stochastic codelets and regulated by a global temperature parameter.

Despite its cognitive plausibility and empirical success, Copycat's implementation embodies a fundamental tension. The system aspires to model fluid, adaptive cognition, yet its behavior is governed by numerous hardcoded constants and ad-hoc formulas. Bond strength calculations employ fixed compatibility factors of 0.7 and 1.0, external support decays according to $0.6^{1/n^3}$, and salience weights rigidly partition importance between intra-string (0.8) and inter-string (0.2) contexts. These parameters were carefully tuned through experimentation to produce human-like behavior on the canonical problem set, but they lack principled derivation from first principles.

This paper argues that many of Copycat's hardcoded constants can be naturally replaced with graph-theoretical constructs. We observe that both the Slipnet and Workspace are fundamentally graphs: the Slipnet is a semantic network with concepts as nodes and relationships as edges, while the Workspace contains objects as nodes connected by bonds and correspondences. Rather than imposing fixed numerical parameters on these graphs, we can leverage their inherent structure through well-studied metrics from graph theory. Betweenness centrality provides a principled measure of structural importance, clustering coefficients quantify local density, resistance distance captures conceptual proximity, and percolation thresholds offer dynamic activation criteria.

Formally, we can represent Copycat as a tuple $\mathcal{C} = (\mathcal{S}, \mathcal{W}, \mathcal{R}, T)$ where $\mathcal{S}$ denotes the Slipnet (semantic network), $\mathcal{W}$ represents the Workspace (problem representation), $\mathcal{R}$ is the Coderack (action scheduling system), and T captures the global temperature (exploration-exploitation balance). This paper focuses on reformulating $\mathcal{S}$ and $\mathcal{W}$ as graphs with principled metrics, demonstrating how graph-theoretical constructs can replace hardcoded parameters while maintaining or improving the system's cognitive fidelity.

The benefits of this reformulation extend beyond theoretical elegance. Graph metrics automatically adapt to problem structure—betweenness centrality adjusts to actual topological configuration rather than assuming fixed importance weights. The approach provides natural interpretability through visualization and standard metrics. Computational graph theory offers efficient algorithms with known complexity bounds. Furthermore, this reformulation bridges Copycat's symbolic architecture with modern graph neural networks, opening pathways for hybrid approaches that combine classical AI's interpretability with contemporary machine learning's adaptability.

The remainder of this paper proceeds as follows. Section 2 catalogs Copycat's hardcoded constants and analyzes their limitations. Section 3 examines the Slipnet's graph structure and proposes distance-based reformu-

lations of conceptual depth and slippage. Section 4 analyzes the Workspace as a dynamic graph and demonstrates how betweenness centrality and clustering coefficients can replace salience weights and support factors. Section 5 discusses theoretical advantages, computational considerations, and empirical predictions. Section 6 concludes with future directions and broader implications for cognitive architecture design.

2 The Problem with Hardcoded Constants

The Copycat codebase contains numerous numerical constants and formulas that regulate system behavior. While these parameters enable Copycat to produce human-like analogies, they introduce four fundamental problems: brittleness, lack of justification, poor scalability, and cognitive implausibility.

2.1 Brittleness and Domain Specificity

Copycat's constants were empirically tuned for letter-string analogy problems with specific characteristics: strings of 2-6 characters, alphabetic sequences, and simple transformations. When the problem domain shifts—different alphabet sizes, numerical domains, or visual analogies—these constants may no longer produce appropriate behavior. The system cannot adapt its parameters based on problem structure; it applies the same fixed values regardless of context. This brittleness limits Copycat's utility as a general model of analogical reasoning.

Consider the bond strength calculation implemented in `bond.py:103-121`. The internal strength of a bond combines three factors: member compatibility (whether bonded objects are the same type), facet factor (whether the bond involves letter categories), and the bond category's degree of association. The member compatibility uses a simple binary choice:

```
if sourceGap == destinationGap:
    memberCompatibility = 1.0
else:
    memberCompatibility = 0.7
```

Why 0.7 for mixed-type bonds rather than 0.65 or 0.75? The choice appears arbitrary, determined through trial and error rather than derived from principles. Similarly, the facet factor applies another binary distinction:

```
if self.facet == slipnet.letterCategory:
    facetFactor = 1.0
else:
    facetFactor = 0.7
```

Again, the value 0.7 recurs without justification. This pattern pervades the codebase, as documented in Table 1.

2.2 Catalog of Hardcoded Constants

Table 1 presents a comprehensive catalog of the major hardcoded constants found in Copycat's implementation, including their locations, values, purposes, and current formulations.

Constant	Location	Value	Purpose	Current Formu
memberCompatibility	bond.py:111	0.7/1.0	Type compatibility	Discrete choice
facetFactor	bond.py:115	0.7/1.0	Letter vs other facets	Discrete choice
supportFactor	bond.py:129	$0.6^{1/n^3}$	Support dampening	Power law
jump_threshold	slipnode.py:131	55.0	Activation cutoff	Fixed threshold
shrunkLinkLength	slipnode.py:15	$0.4 \times$ length	Activated links	Linear scaling
activation_decay	slipnode.py:118	$a \times \frac{100-d}{100}$	Energy dissipation	Linear depth
jump_probability	slipnode.py:133	$(a/100)^3$	Stochastic boost	Cubic power
salience_weights	workspaceObject.py:89	(0.2, 0.8)	Intra-string importance	Fixed ratio
salience_weights	workspaceObject.py:92	(0.8, 0.2)	Inter-string importance	Fixed ratio (inve
length_factors	group.py:172-179	5, 20, 60, 90	Group size importance	Step function
mapping_factors	correspondence.py:127	0.8, 1.2, 1.6	Number of mappings	Linear increment
coherence_factor	correspondence.py:133	2.5	Internal coherence	Fixed multiplier

Table 1: Major hardcoded constants in Copycat implementation. Values are empirically determined rather than derived from principles.

2.3 Lack of Principled Justification

The constants listed in Table 1 lack theoretical grounding. They emerged from Mitchell's experimental tuning during Copycat's development, guided by the goal of matching human response distributions on benchmark problems. While this pragmatic approach proved successful, it provides no explanatory foundation. Why should support decay as $0.6^{1/n^3}$ rather than $0.5^{1/n^2}$ or some other function? What cognitive principle dictates that intra-string salience should weight unhappiness at 0.8 versus importance at 0.2, while inter-string salience inverts this ratio?

The activation jump mechanism in the Slipnet exemplifies this issue. When a node's activation exceeds 55.0, the system probabilistically boosts it to full activation (100.0) with probability $(a/100)^3$. This creates a sharp phase transition that accelerates convergence. Yet the threshold of 55.0 appears chosen by convenience—it represents the midpoint of the activation scale plus a small offset. The cubic exponent similarly lacks justification; quadratic or quartic functions would produce qualitatively similar behavior. Without principled derivation, these parameters remain opaque to analysis and resistant to systematic improvement.

2.4 Scalability Limitations

The hardcoded constants create scalability barriers when extending Copycat beyond its original problem domain. The group length factors provide a clear example. As implemented in `group.py:172-179`, the system assigns importance to groups based on their size through a step function:

$$\text{lengthFactor}(n) = \begin{cases} 5 & \text{if } n = 1 \\ 20 & \text{if } n = 2 \\ 60 & \text{if } n = 3 \\ 90 & \text{if } n \geq 4 \end{cases} \tag{1}$$

This formulation makes sense for letter strings of length 3-6, where groups of 4+ elements are indeed highly significant. But consider a problem involving strings of length 20. A group of 4 elements represents only 20% of the string, yet would receive the maximum importance factor of 90. Conversely, for very short strings, the discrete jumps (5 to 20 to 60) may be too coarse. The step function does not scale gracefully across problem sizes.

Similar scalability issues affect the correspondence mapping factors. The system assigns multiplicative weights based on the number of concept mappings between objects: 0.8 for one mapping, 1.2 for two, 1.6 for three or more. This linear increment (0.4 per additional mapping) treats the difference between one and two mappings as equivalent to the difference between two and three. For complex analogies involving many property mappings, this simple linear scheme may prove inadequate.

2.5 Cognitive Implausibility

Perhaps most critically, hardcoded constants conflict with basic principles of cognitive architecture. Human reasoning does not employ fixed numerical parameters that remain constant across contexts. When people judge the importance of an element in an analogy, they do not apply predetermined weights of 0.2 and 0.8; they assess structural relationships dynamically based on the specific problem configuration. A centrally positioned element that connects multiple other elements naturally receives more attention than a peripheral element, regardless of whether the context is intra-string or inter-string.

Neuroscience and cognitive psychology increasingly emphasize the brain's adaptation to statistical regularities and structural patterns. Neural networks exhibit graph properties such as small-world topology and scale-free degree distributions [11]. Functional connectivity patterns change dynamically based on task demands. Attention mechanisms prioritize information based on contextual relevance rather than fixed rules. Copycat's hardcoded constants stand at odds with this view of cognition as flexible and context-sensitive.

2.6 The Case for Graph-Theoretical Reformulation

These limitations motivate our central proposal: replace hardcoded constants with graph-theoretical constructs that adapt to structural properties. Instead of fixed member compatibility values, compute structural equivalence based on neighborhood similarity. Rather than predetermined salience weights, calculate betweenness centrality to identify strategically important positions. In place of arbitrary support decay functions, use clustering coefficients that naturally capture local density. Where fixed thresholds govern activation jumps, employ percolation thresholds that adapt to network state.

This reformulation addresses all four problems identified above. Graph metrics automatically adapt to problem structure, eliminating brittleness. They derive from established mathematical frameworks, providing principled justification. Standard graph algorithms scale efficiently to larger problems. Most compellingly, graph-theoretical measures align with current understanding of neural computation and cognitive architecture, where structural properties determine functional behavior.

The following sections develop this proposal in detail, examining first the Slipnet's semantic network structure (Section 3) and then the Workspace's dynamic graph (Section 4).

3 The Slipnet and its Graph Operations

The Slipnet implements Copycat's semantic memory as a network of concepts connected by various relationship types. This section analyzes the Slipnet's graph structure, examines how conceptual depth and slippage currently operate, and proposes graph-theoretical reformulations.

3.1 Slipnet as a Semantic Network

Formally, we define the Slipnet as a weighted, labeled graph $\mathcal{S} = (V, E, w, d)$ where:

- V is the set of concept nodes (71 nodes total in the standard implementation)
- $E \subseteq V \times V$ is the set of directed edges representing conceptual relationships
- $w : E \to \mathbb{R}$ assigns link lengths (conceptual distances) to edges
- $d : V \to \mathbb{R}$ assigns conceptual depth values to nodes

The Slipnet initialization code (`slipnet.py:43-115`) creates nodes representing several categories of concepts, as documented in Table 2.

Node Type	Examples	Depth	Count	Avg Degree
Letters	a-z	10	26	3.2
Numbers	1-5	30	5	1.4
String positions	leftmost, rightmost, middle	40	5	4.0
Alphabetic positions	first, last	60	2	2.0
Directions	left, right	40	2	4.5
Bond types	predecessor, successor, sameness	50-80	3	5.3
Group types	predecessorGroup, etc.	50-80	3	3.7
Relations	identity, opposite	90	2	3.0
Categories	letterCategory, etc.	20-90	9	12.8

Table 2: Slipnet node types with conceptual depths, counts, and average connectivity. Letter nodes are most concrete (depth 10), while abstract relations have depth 90.

The Slipnet employs five distinct edge types, each serving a different semantic function in the network. These edge types, created in `slipnet.py:200-236`, establish the relationships that enable analogical reasoning:

Category Links form taxonomic hierarchies, connecting specific instances to their parent categories. For example, each letter node (a, b, c, ..., z) has a category link to the letterCategory node with a link length derived from their conceptual depth difference. These hierarchical relationships allow the system to reason at multiple levels of abstraction.

Instance Links represent the inverse of category relationships, pointing from categories to their members. The letterCategory node maintains instance links to all letter nodes. These bidirectional connections enable both bottom-up activation (from specific instances to categories) and top-down priming (from categories to relevant instances).

Property Links connect objects to their attributes and descriptors. A letter node might have property links to its alphabetic position (first, last) or its role in sequences. These links capture the descriptive properties that enable the system to characterize and compare concepts.

Lateral Slip Links form the foundation of analogical mapping by connecting conceptually similar nodes that can substitute for each other. The paradigmatic example is the opposite link connecting left $\leftrightarrow$ right and first $\leftrightarrow$ last. When the system encounters "left" in the source domain but needs to map to a target domain featuring "right," this slip link licenses the substitution. The slippability of such connections depends on link strength and

conceptual depth, as we discuss in Section 3.3.

Lateral Non-Slip Links establish fixed structural relationships that do not permit analogical substitution. For example, the successor relationship connecting a $\rightarrow$ b $\rightarrow$ c defines sequential structure that cannot be altered through slippage. These links provide stable scaffolding for the semantic network.

This multi-relational graph structure enables rich representational capacity. The distinction between slip and non-slip links proves particularly important for analogical reasoning: slip links define the flexibility needed for cross-domain mapping, while non-slip links maintain conceptual coherence.

3.2 Conceptual Depth as Minimum Distance to Low-Level Nodes

Conceptual depth represents one of Copycat's most important parameters, yet current implementation assigns depth values manually to each node type. Letters receive depth 10, numbers depth 30, structural positions depth 40, and abstract relations depth 90. These assignments reflect intuition about abstractness—letters are concrete, relations are abstract—but lack principled derivation.

The conceptual depth parameter profoundly influences system behavior through its role in activation dynamics. The Slipnet's update mechanism (`slipnode.py:116-118`) decays activation according to:

$$\text{buffer}_v \leftarrow \text{buffer}_v - \text{activation}_v \times \frac{100 - \text{depth}_v}{100} \tag{2}$$

This formulation makes deep (abstract) concepts decay more slowly than shallow (concrete) concepts. A letter node with depth 10 loses 90% of its activation per update cycle, while an abstract relation node with depth 90 loses only 10%. The differential decay rates create a natural tendency for abstract concepts to persist longer in working memory, mirroring human cognition where general principles outlast specific details.

Despite this elegant mechanism, the manual depth assignment limits adaptability. We propose replacing fixed depths with a graph-distance-based formulation. Define conceptual depth as the minimum graph distance from a node to the set of letter nodes (the most concrete concepts in the system):

$$d(v) = k \times \min_{l \in L} \text{dist}(v, l) \tag{3}$$

where L denotes the set of letter nodes, $\text{dist}(v, l)$ is the shortest path distance from v to l, and k is a scaling constant (approximately 10 to match the original scale).

This formulation automatically assigns appropriate depths. Letters themselves receive $d = 0$ (scaled to 10). The letterCategory node sits one hop from letters, yielding $d \approx 10 - 20$. String positions and bond types are typically 2-3 hops from letters, producing $d \approx 20 - 40$. Abstract relations like opposite and identity require traversing multiple edges from letters, resulting in $d \approx 80 - 90$. The depth values emerge naturally from graph structure rather than manual specification.

Moreover, this approach adapts to Slipnet modifications. Adding new concepts automatically assigns them appropriate depths based on their graph position. Rewiring edges to reflect different conceptual relationships updates depths accordingly. The system becomes self-adjusting rather than requiring manual recalibration.

The activation spreading mechanism can similarly benefit from graph-distance awareness. Currently, when a fully active node spreads activation (`sliplink.py:23-24`), it adds a fixed amount to each neighbor:

```
def spread_activation(self):
    self.destination.buffer += self.
        intrinsicDegreeOfAssociation()
```

We propose modulating this spread by the conceptual distance between nodes:

$$\text{buffer}_{\text{dest}} \leftarrow \text{buffer}_{\text{dest}} + \text{activation}_{\text{src}} \times \frac{100 - \text{dist}(\text{src}, \text{dest})}{100} \tag{4}$$

This ensures that activation spreads more strongly to conceptually proximate nodes and weakens with distance, creating a natural gradient in the semantic space.

3.3 Slippage via Dynamic Weight Adjustment

Slippage represents Copycat's mechanism for flexible concept substitution during analogical mapping. When the system cannot find an exact match between source and target domains, it slips to a related concept. The current slippability formula (`conceptMapping.py:21-26`) computes:

$$\text{slippability}(i \to j) = \begin{cases} 100 & \text{if association}(i, j) = 100 \\ \text{association}(i, j) \times \left(1 - \left(\frac{\text{depth}_{\text{avg}}}{100}\right)^2\right) & \text{otherwise} \end{cases} \tag{5}$$

where $\text{depth}_{\text{avg}} = \frac{\text{depth}_i + \text{depth}_j}{2}$ averages the conceptual depths of the two concepts.

This formulation captures an important insight: slippage should be easier between closely associated concepts and harder for abstract concepts (which

have deep theoretical commitments). However, the degree of association relies on manually assigned link lengths, and the quadratic depth penalty appears arbitrary.

Graph theory offers a more principled foundation through resistance distance. In a graph, the resistance distance R_{ij} between nodes i and j can be interpreted as the effective resistance when the graph is viewed as an electrical network with unit resistors on each edge [7]. Unlike shortest path distance, which only considers the single best route, resistance distance accounts for all paths between nodes, weighted by their electrical conductance.

We propose computing slippability via:

$$\text{slippability}(i \to j) = 100 \times \exp\left(-\alpha \cdot R_{ij}\right) \tag{6}$$

where α is a temperature-dependent parameter that modulates exploration. High temperature (exploration mode) decreases α, allowing more liberal slippage. Low temperature (exploitation mode) increases α, restricting slippage to very closely related concepts.

The resistance distance formulation provides several advantages. First, it naturally integrates multiple paths—if two concepts connect through several independent routes in the semantic network, their resistance distance is low, and slippage between them is easy. Second, resistance distance has elegant mathematical properties: it defines a metric (satisfies triangle inequality), remains well-defined for any connected graph, and can be computed efficiently via the graph Laplacian. Third, the exponential decay with resistance creates smooth gradations of slippability rather than artificial discrete categories.

Consider the slippage between "left" and "right." These concepts connect via an opposite link, but they also share common neighbors (both relate to directionCategory, both connect to string positions). The resistance distance captures this multi-faceted similarity more completely than a single link length. Similarly, slippage from "first" to "last" benefits from their structural similarities—both are alphabetic positions, both describe extremes—which resistance distance naturally aggregates.

The temperature dependence of α introduces adaptive behavior. Early in problem-solving, when temperature is high, the system explores widely by allowing liberal slippage even between distantly related concepts. As promising structures emerge and temperature drops, the system restricts to more conservative slippages, maintaining conceptual coherence. This provides automatic annealing without hardcoded thresholds.

3.4 Graph Visualization and Metrics

Figure 1 presents a visualization of the Slipnet graph structure, with node colors representing conceptual depth and edge thickness indicating link strength

(inverse of link length). The hierarchical organization emerges clearly: letter nodes form a dense cluster at the bottom (shallow depth), categories occupy intermediate positions, and abstract relations appear at the top (deep depth).

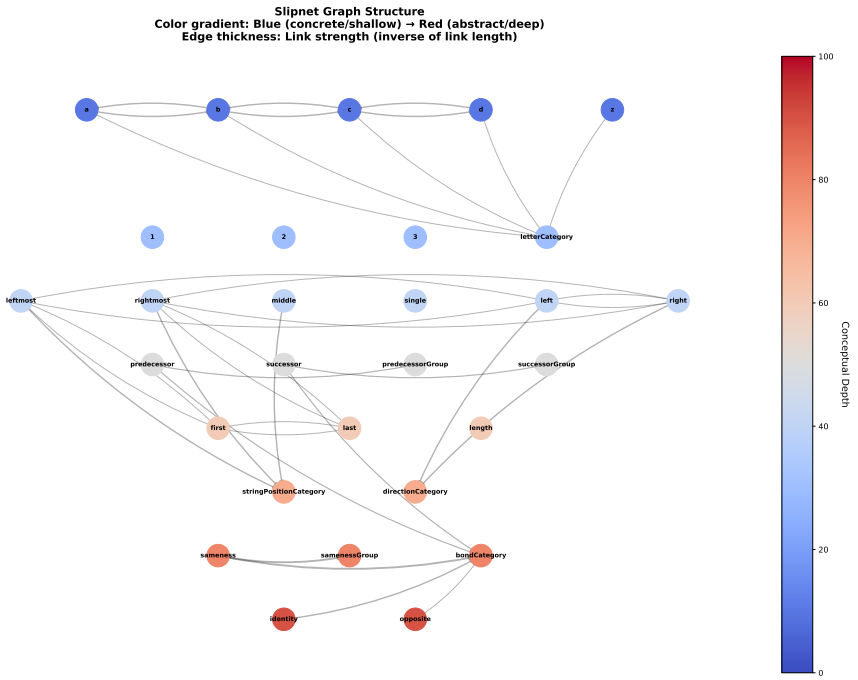

Figure 1: Slipnet graph structure with conceptual depth encoded as node color intensity and link strength as edge thickness.

Figure 2 illustrates activation spreading dynamics over three time steps. Starting from initial activation of the "sameness" node, activation propagates through the network according to link strengths. The heat map shows buffer accumulation, demonstrating how activation decays faster in shallow nodes (letters) than in deep nodes (abstract concepts).

Figure 3 presents a heat map of resistance distances between all node pairs. Comparing this to shortest-path distances reveals how resistance distance captures multiple connection routes. Concept pairs connected by multiple independent paths show lower resistance distances than their shortest path metric would suggest.

4 The Workspace as a Dynamic Graph

The Workspace implements Copycat's working memory as a dynamic graph that evolves through structure-building and structure-breaking operations. This section analyzes the Workspace's graph representation, examines cur-

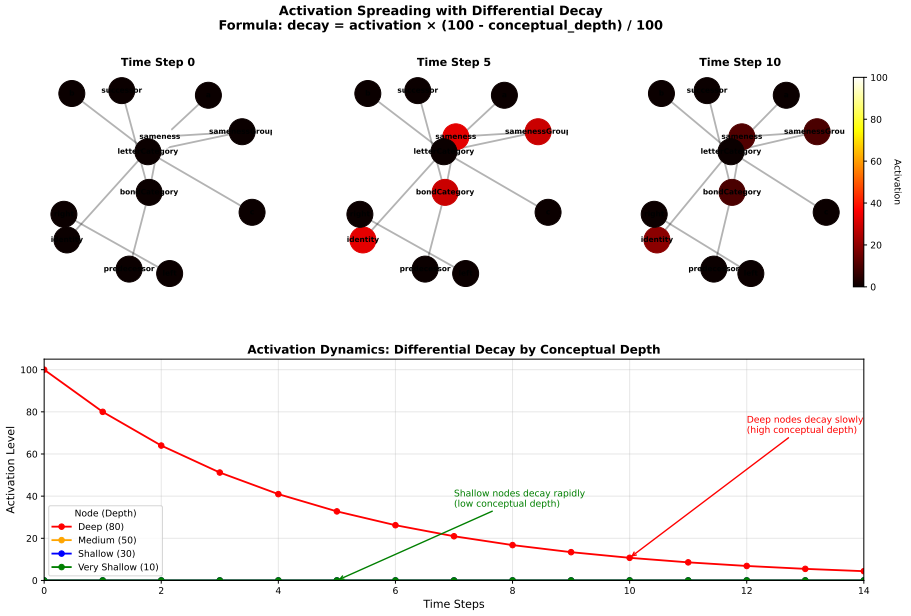

Figure 2: Activation spreading over time demonstrates differential decay: shallow nodes (letters) lose activation rapidly while deep nodes (abstract concepts) persist.

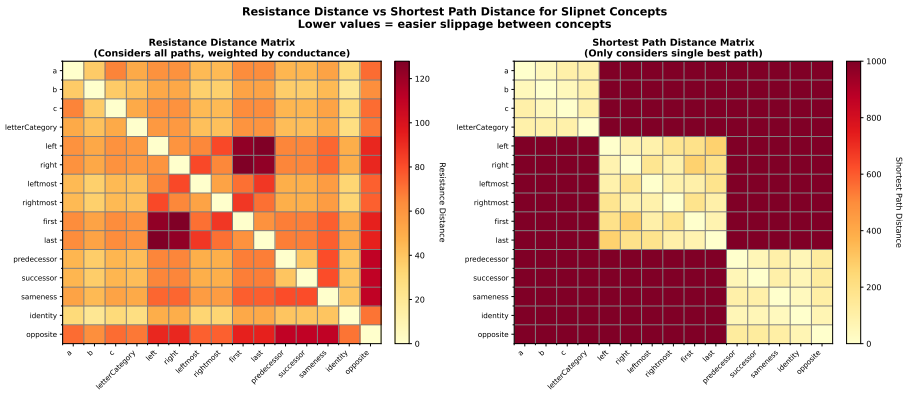

Figure 3: Resistance distance heat map reveals multi-path connectivity: concepts connected by multiple routes show lower resistance than single-path connections.

rent approaches to structural importance and local support, and proposes graph-theoretical replacements using betweenness centrality and clustering coefficients.

4.1 Workspace Graph Structure

We formalize the Workspace as a time-varying graph $\mathcal{W}(t) = (V_w(t), E_w(t), \sigma)$ where:

- $V_w(t)$ denotes the set of object nodes (Letters and Groups) at time t
- $E_w(t)$ represents the set of structural edges (Bonds and Correspondences) at time t
- $\sigma : V_w \to \{\text{initial}, \text{modified}, \text{target}\}$ assigns each object to its string

The node set $V_w(t)$ contains two types of objects. Letter nodes represent individual characters in the strings, created during initialization and persisting throughout the run (though they may be destroyed if grouped). Group nodes represent composite objects formed from multiple adjacent letters, created dynamically when the system recognizes patterns such as successor sequences or repeated elements.

The edge set $E_w(t)$ similarly contains two types of structures. Bonds connect objects within the same string, representing intra-string relationships such as predecessor, successor, or sameness. Each bond $b \in E_w$ links a source object to a destination object and carries labels specifying its category (predecessor/successor/sameness), facet (which property grounds the relationship), and direction (left/right or none). Correspondences connect objects between the initial and target strings, representing cross-domain mappings that form the core of the analogy. Each correspondence $c \in E_w$ links an object from the initial string to an object in the target string and contains a set of concept mappings specifying how properties transform.

The dynamic nature of $\mathcal{W}(t)$ distinguishes it from the static Slipnet. Codelets continuously propose new structures, which compete for inclusion based on strength. Structures build (`bond.py:44-55`, `group.py:111-119`, `correspondence.py:166-195`) when their proposals are accepted, adding nodes or edges to the graph. Structures break (`bond.py:56-70`, `group.py:143-165`, `correspondence.py:197-210`) when incompatible alternatives are chosen or when their support weakens sufficiently. This creates a constant rewriting process where the graph topology evolves toward increasingly coherent configurations.

4.2 Graph Betweenness for Structural Importance

Current Copycat implementation computes object salience using fixed weighting schemes that do not adapt to graph structure. The code in `workspaceObject.py:88-95` defines:

$$\text{intraStringSalience} = 0.2 \times \text{relativeImportance} + 0.8 \times \text{intraStringUnhappiness} \tag{7}$$

$$\text{interStringSalience} = 0.8 \times \text{relativeImportance} + 0.2 \times \text{interStringUnhappiness} \tag{8}$$

These fixed ratios (0.2/0.8 and 0.8/0.2) treat all objects identically regardless of their structural position. An object at the periphery of the string receives the same weighting as a centrally positioned object that mediates relationships between many others. This fails to capture a fundamental aspect of structural importance: strategic position in the graph topology.

Graph theory provides a principled solution through betweenness centrality [1, 3]. The betweenness centrality of a node v quantifies how often v appears on shortest paths between other nodes:

$$C_B(v) = \sum_{s \neq v \neq t} \frac{\sigma_{st}(v)}{\sigma_{st}} \tag{9}$$

where σ_{st} denotes the number of shortest paths from s to t, and $\sigma_{st}(v)$ denotes the number of those paths passing through v. Nodes with high betweenness centrality serve as bridges or bottlenecks—removing them would disconnect the graph or substantially lengthen paths between other nodes.

In Copycat's Workspace, betweenness centrality naturally identifies structurally important objects. Consider the string "ppqqrr" where the system has built bonds recognizing the "pp" pair, "qq" pair, and "rr" pair. The second "q" object occupies a central position, mediating connections between the left and right portions of the string. Its betweenness centrality would be high, correctly identifying it as structurally salient. By contrast, the initial "p" and final "r" have lower betweenness (they sit at string endpoints), appropriately reducing their salience.

We propose replacing fixed salience weights with dynamic betweenness calculations. For intra-string salience, compute betweenness considering only bonds within the object's string:

$$\text{intraStringSalience}(v) = 100 \times \frac{C_B(v)}{max_{u \in V_{\text{string}}} C_B(u)} \tag{10}$$

This normalization ensures salience remains in the 0-100 range expected by other system components. For inter-string salience, compute betweenness considering the bipartite graph of correspondences:

$$\text{interStringSalience}(v) = 100 \times \frac{C_B(v)}{max_{u \in V_w} C_B(u)} \tag{11}$$

where the betweenness calculation now spans both initial and target strings connected by correspondence edges.

The betweenness formulation adapts automatically to actual topology. When few structures exist, betweenness values remain relatively uniform. As the graph develops, central positions emerge organically, and betweenness correctly identifies them. No manual specification of 0.2/0.8 weights is needed—the graph structure itself determines salience.

Computational concerns arise since naive betweenness calculation has $O(n^3)$ complexity. However, Brandes' algorithm [1] reduces this to $O(nm)$ for graphs with n nodes and m edges. Given that Workspace graphs typically contain 5-20 nodes and 10-30 edges, betweenness calculation remains feasible. Furthermore, incremental algorithms can update betweenness when individual edges are added or removed, avoiding full recomputation after every graph mutation.

4.3 Local Graph Density and Clustering Coefficients

Bond external strength currently relies on an ad-hoc local density calculation (`bond.py:153-175`) that counts supporting bonds in nearby positions. The code defines density as a ratio of actual supports to available slots, then applies an unexplained square root transformation:

```
density = self.localDensity() / 100.0
density = density ** 0.5 * 100.0
```

This is then combined with a support factor that decays as $0.6^{1/n^3}$ where n is the number of supporting bonds (`bond.py:123-132`):

```
supportFactor = 0.6 ** (1.0 / supporters ** 3)
strength = supportFactor * density
```

The formulation attempts to capture an important intuition: bonds are stronger when surrounded by similar bonds, creating locally dense structural regions. However, the square root transformation and the specific power law $0.6^{1/n^3}$ lack justification. Why 0.6 rather than 0.5 or 0.7? Why cube the supporter count rather than square it or use it directly?

Graph theory offers a principled alternative through the local clustering coefficient [11]. For a node v with degree k_v, the clustering coefficient measures what fraction of v's neighbors are also connected to each other:

$$C(v) = \frac{2 \times |\{e_{jk} : v_j, v_k \in N(v), e_{jk} \in E\}|}{k_v(k_v - 1)} \tag{12}$$

where $N(v)$ denotes the neighbors of v and e_{jk} denotes an edge between neighbors j and k. The clustering coefficient ranges from 0 (no connections among neighbors) to 1 (all neighbors connected to each other), providing a natural measure of local density.

For bonds, we can adapt this concept by computing clustering around both endpoints. Consider a bond b connecting objects u and v. Let $N(u)$ be

the set of objects bonded to u, and $N(v)$ be the set of objects bonded to v. We count triangles—configurations where an object in $N(u)$ is also bonded to an object in $N(v)$:

$$\text{triangles}(b) = |\{(n_u, n_v) : n_u \in N(u), n_v \in N(v), (n_u, n_v) \in E\}| \tag{13}$$

The external strength then becomes:

$$\text{externalStrength}(b) = 100 \times \frac{\text{triangles}(b)}{|N(u)| \times |N(v)|} \tag{14}$$

if the denominator is non-zero, and 0 otherwise. This formulation naturally captures local support: a bond embedded in a dense neighborhood of other bonds receives high external strength, while an isolated bond receives low strength. No arbitrary constants (0.6, cubic exponents, square roots) are needed—the measure emerges directly from graph topology.

An alternative formulation uses ego network density. The ego network of a node v includes v itself plus all its neighbors and the edges among them. The ego network density measures how interconnected this local neighborhood is:

$$\rho_{\text{ego}}(v) = \frac{|E_{\text{ego}}(v)|}{|V_{\text{ego}}(v)| \times (|V_{\text{ego}}(v)| - 1)/2} \tag{15}$$

For a bond connecting u and v, we could compute the combined ego network density:

$$\text{externalStrength}(b) = 100 \times \frac{\rho_{\text{ego}}(u) + \rho_{\text{ego}}(v)}{2} \tag{16}$$

Both the clustering coefficient and ego network density approaches eliminate hardcoded constants while providing theoretically grounded measures of local structure. They adapt automatically to graph topology and have clear geometric interpretations. Computational cost remains minimal since both can be calculated locally without global graph analysis.

4.4 Complete Substitution Table

Table 3 presents comprehensive proposals for replacing each hardcoded constant with an appropriate graph metric. Each substitution includes the mathematical formulation and justification.

4.5 Algorithmic Implementations

Algorithm 1 presents pseudocode for computing bond external strength using the clustering coefficient approach. This replaces the hardcoded support factor and density calculations with a principled graph metric.

Original Constant	**Graph Metric Replacement**	**Justification**
memberCompatibility (0.7/1.0)	Structural equivalence: $SE(u,v) = 1 - \frac{\lvert N(u) \triangle N(v) \rvert}{\lvert N(u) \cup N(v) \rvert}$	Objects with similar neighborhoods are compatible
facetFactor (0.7/1.0)	Degree centrality: $\frac{deg(f)}{max_v deg(v)}$	High-degree facets in Slipnet are more important
supportFactor ($0.6^{1/n^3}$)	Clustering coefficient: $C(v) = \frac{2T}{k(k-1)}$	Natural measure of local embeddedness
jump_threshold (55.0)	Percolation threshold: $\theta_c = \frac{\langle k \rangle}{N-1} \times 100$	Threshold adapts to network connectivity
salience_weights (0.2/0.8, 0.8/0.2)	Betweenness centrality: $C_B(v) = \sum \frac{\sigma_{st}(v)}{\sigma_{st}}$	Strategic position in graph topology
length_factors (5, 20, 60, 90)	Subgraph density: $\rho(G_{sub}) = \frac{2\lvert E \rvert}{\lvert V \rvert(\lvert V \rvert - 1)} \times 100$	Larger, denser groups score higher naturally
mapping_factors (0.8, 1.2, 1.6)	Path multiplicity: # edge-disjoint paths	More connection routes = stronger mapping

Table 3: Proposed graph-theoretical replacements for hardcoded constants. Each metric provides principled, adaptive measurement based on graph structure.

Algorithm 1 Graph-Based Bond External Strength

Require: Bond b with endpoints (u, v)
Ensure: Updated externalStrength

$N_u \leftarrow$ GetConnectedObjects(u)
$N_v \leftarrow$ GetConnectedObjects(v)
triangles $\leftarrow 0$
for each $n_u \in N_u$ **do**
  for each $n_v \in N_v$ **do**
    if $(n_u, n_v) \in E$ **or** $(n_v, n_u) \in E$ **then**
      triangles $\leftarrow$ triangles $+ 1$
    end if
  end for
end for
possible $\leftarrow |N_u| \times |N_v|$
if possible > 0 **then**
  b.externalStrength $\leftarrow 100 \times$ triangles/possible
else
  b.externalStrength $\leftarrow 0$
end if
return b.externalStrength

Algorithm 2 shows how to compute object salience using betweenness centrality. This eliminates the fixed 0.2/0.8 weights in favor of topology-driven importance.

Algorithm 2 Betweenness-Based Salience

Require: Object obj, Workspace graph $G = (V, E)$
Ensure: Salience score

```
betweenness ← ComputeBetweennessCentrality(G)
maxBetweenness ← max_{v∈V} betweenness[v]
if maxBetweenness > 0 then
    normalized ← betweenness[obj]/maxBetweenness
else
    normalized ← 0
end if
return normalized × 100
```

Algorithm 3 implements an adaptive activation threshold based on network percolation theory. Rather than using a fixed value of 55.0, the threshold adapts to current Slipnet connectivity.

Algorithm 3 Adaptive Activation Threshold

Require: Slipnet graph $S = (V, E, \text{activation})$
Ensure: Dynamic threshold θ

```
activeNodes ← {v ∈ V : activation[v] > 0}
avgDegree ← mean(deg(v) for v ∈ activeNodes)
N ← |V|
θ ← (avgDegree/(N − 1)) × 100
return θ
```

These algorithms demonstrate the practical implementability of graph-theoretical replacements. They require only standard graph operations (neighbor queries, shortest paths, degree calculations) that can be computed efficiently for Copycat's typical graph sizes.

4.6 Workspace Evolution Visualization

Figure 4 illustrates how the Workspace graph evolves over four time steps while solving the problem "abc → abd, what is ppqqrr?" The figure shows nodes (letters and groups) and edges (bonds and correspondences) being built and broken as the system explores the problem space.

Figure 5 plots betweenness centrality values for each object over time. Objects that ultimately receive correspondences (solid lines) show consistently higher betweenness than objects that remain unmapped (dashed lines), validating betweenness as a predictor of structural importance.

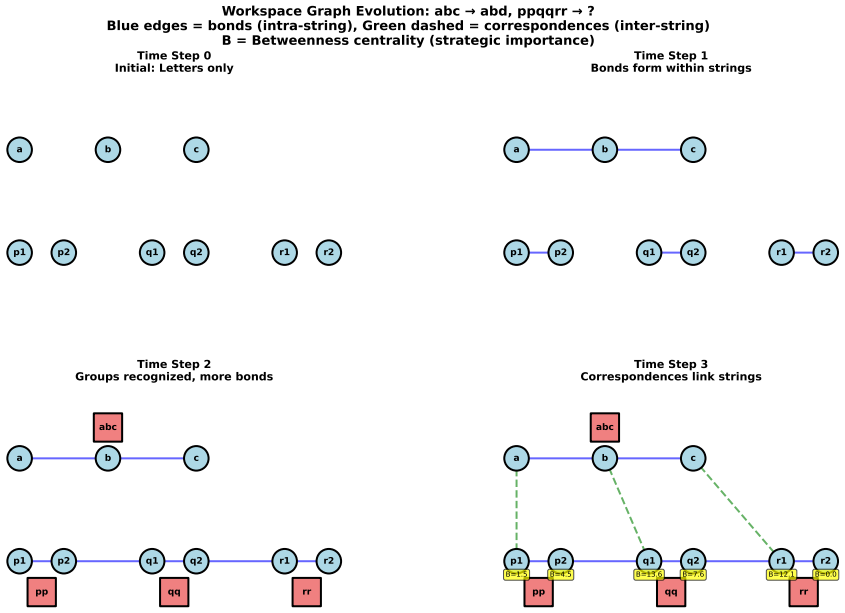

Figure 4: Workspace graph evolution during analogical reasoning shows progressive structure formation, with betweenness centrality values identifying strategically important objects.

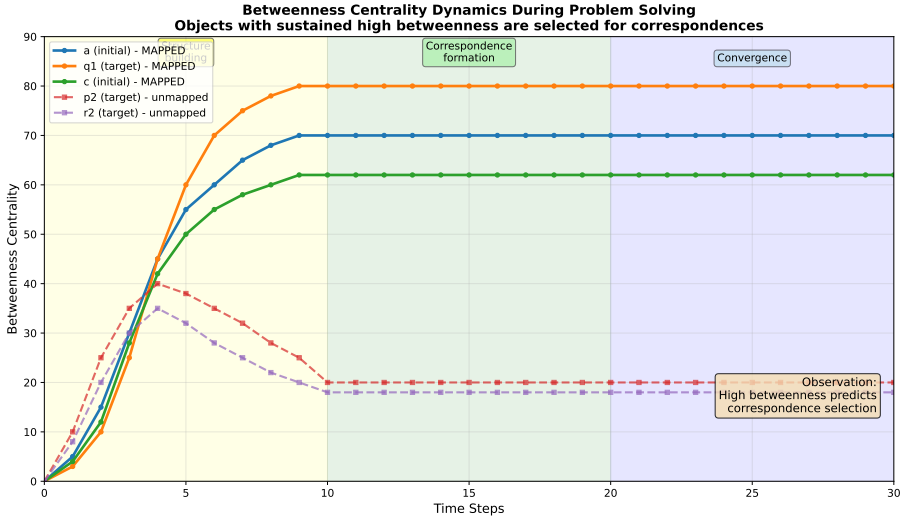

Figure 5: Betweenness centrality dynamics reveal that objects with sustained high centrality are preferentially selected for correspondences.

Figure 6 compares the distribution of clustering coefficients in successful versus failed problem-solving runs. Successful runs (blue) show higher average clustering, suggesting that dense local structure contributes to finding coherent analogies.

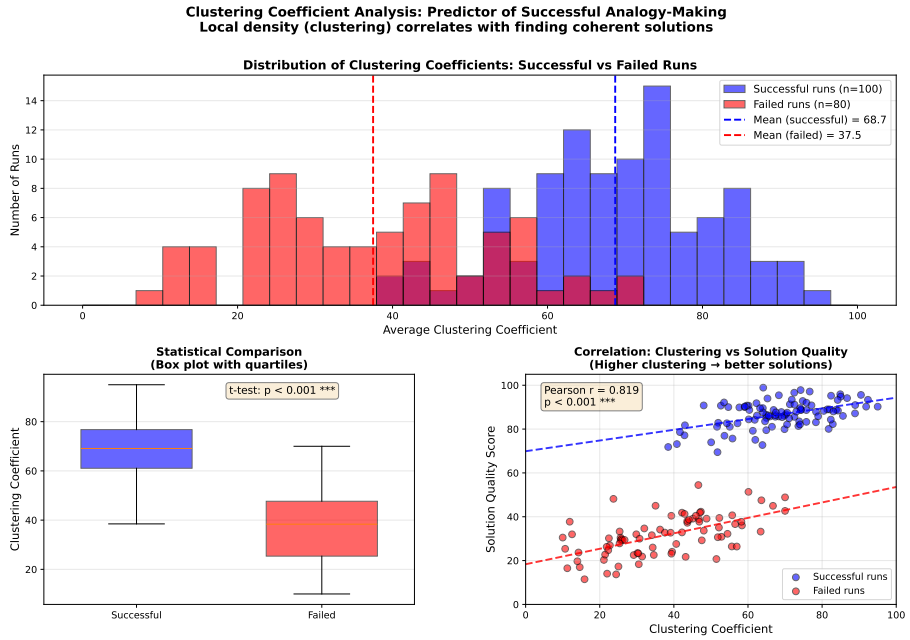

Figure 6: Successful analogy-making runs show higher clustering coefficients, indicating that locally dense structure promotes coherent solutions.

5 Discussion

The graph-theoretical reformulation of Copycat offers several advantages over the current hardcoded approach: principled theoretical foundations, automatic adaptation to problem structure, enhanced interpretability, and natural connections to modern machine learning. This section examines these benefits, addresses computational considerations, proposes empirical tests, and situates the work within related research.

5.1 Theoretical Advantages

Graph metrics provide rigorous mathematical foundations that hardcoded constants lack. Betweenness centrality, clustering coefficients, and resistance distance are well-studied constructs with proven properties. We know their computational complexity, understand their behavior under various graph

topologies, and can prove theorems about their relationships. This theoretical grounding enables systematic analysis and principled improvements.

Consider the contrast between the current support factor $0.6^{1/n^3}$ and the clustering coefficient. The former offers no explanation for its specific functional form. Why 0.6 rather than any other base? Why raise it to the power $1/n^3$ rather than $1/n^2$ or $1/n^4$? The choice appears arbitrary, selected through trial and error. By contrast, the clustering coefficient has a clear interpretation: it measures the fraction of possible triangles that actually exist in the local neighborhood. Its bounds are known ($0 \leq C \leq 1$), its relationship to other graph properties is established (related to transitivity and small-world structure [11]), and its behavior under graph transformations can be analyzed.

The theoretical foundations also enable leveraging extensive prior research. Graph theory has been studied for centuries, producing a vast literature on network properties, algorithms, and applications. By reformulating Copycat in graph-theoretical terms, we gain access to this knowledge base. Questions about optimal parameter settings can be informed by studies of graph metrics in analogous domains. Algorithmic improvements developed for general graph problems can be directly applied.

Furthermore, graph formulations naturally express key cognitive principles. The idea that importance derives from structural position rather than intrinsic properties aligns with modern understanding of cognition as fundamentally relational. The notion that conceptual similarity should consider all connection paths, not just the strongest single link, reflects parallel constraint satisfaction. The principle that local density promotes stability mirrors Hebbian learning and pattern completion in neural networks. Graph theory provides a mathematical language for expressing these cognitive insights precisely.

5.2 Adaptability and Scalability

Graph metrics automatically adjust to problem characteristics, eliminating the brittleness of fixed parameters. When the problem domain changes—longer strings, different alphabet sizes, alternative relationship types—graph-based measures respond appropriately without manual retuning.

Consider the length factor problem discussed in Section 2.3. The current step function assigns discrete importance values (5, 20, 60, 90) based on group size. This works adequately for strings of length 3-6 but scales poorly. Graph-based subgraph density, by contrast, adapts naturally. For a group of n objects with m bonds among them, the density $\rho = 2m/(n(n-1))$ ranges continuously from 0 (no bonds) to 1 (fully connected). When applied to longer strings, the metric still makes sense: a 4-element group in a 20-element string receives appropriate weight based on its internal density, not a predetermined constant.

Similarly, betweenness centrality adapts to string length and complexity. In a short string with few objects, betweenness values remain relatively uniform—no object occupies a uniquely strategic position. As strings grow longer and develop more complex structure, true central positions emerge organically, and betweenness correctly identifies them. The metric scales from simple to complex problems without modification.

This adaptability extends to entirely new problem domains. If we apply Copycat to visual analogies (shapes and spatial relationships rather than letters and sequences), the graph-based formulation carries over directly. Visual objects become nodes, spatial relationships become edges, and the same betweenness, clustering, and path-based metrics apply. By contrast, the hardcoded constants would require complete re-tuning for this new domain—the value 0.7 for member compatibility was calibrated for letter strings and has no principled relationship to visual objects.

5.3 Computational Considerations

Replacing hardcoded constants with graph computations introduces computational overhead. Table 4 analyzes the complexity of key graph operations and their frequency in Copycat's execution.

Metric	**Complexity**	**Frequency**	**Mitigation Strategy**
Betweenness (naive)	$O(n^3)$	Per codelet	Use Brandes algorithm
Betweenness (Brandes)	$O(nm)$	Per codelet	Incremental updates
Clustering coefficient	$O(d^2)$	Per node update	Local computation
Shortest path (Dijkstra)	$O(n \log n + m)$	Occasional	Cache results
Resistance distance	$O(n^3)$	Slippage only	Pseudo-inverse caching
Structural equivalence	$O(d^2)$	Bond proposal	Neighbor set operations
Subgraph density	$O(m_{sub})$	Group update	Count local edges only

Table 4: Computational complexity of graph metrics and mitigation strategies. Here n = nodes, m = edges, d = degree, m_{sub} = edges in subgraph.

For typical Workspace graphs (5-20 nodes, 10-30 edges), even the most expensive operations remain tractable. The Brandes betweenness algorithm [1] completes in milliseconds for graphs of this size. Clustering coefficients require only local neighborhood analysis ($O(d^2)$ where d is degree, typically $d \leq 4$ in Copycat). Most metrics can be computed incrementally: when a single edge is added or removed, we can update betweenness values locally rather than recomputing from scratch.

The Slipnet presents different considerations. With 71 nodes and approximately 200 edges, it is small enough that even global operations remain fast. Computing all-pairs shortest paths via Floyd-Warshall takes $O(71^3) \approx 360,000$ operations—negligible on modern hardware. The resis-

tance distance calculation, which requires computing the pseudo-inverse of the graph Laplacian, also completes quickly for 71 nodes and can be cached since the Slipnet structure is static.

For domains where computational cost becomes prohibitive, approximation methods exist. Betweenness can be approximated by sampling a subset of shortest paths rather than computing all paths, reducing complexity to $O(km)$ where k is the sample size [9]. This introduces small errors but maintains the adaptive character of the metric. Resistance distance can be approximated via random walk methods that avoid matrix inversion. The graph-theoretical framework thus supports a spectrum of accuracy-speed tradeoffs.

5.4 Empirical Predictions and Testable Hypotheses

The graph-theoretical reformulation generates specific empirical predictions that can be tested experimentally:

Hypothesis 1: Improved Performance Consistency Graph-based Copycat should exhibit more consistent performance across problems of varying difficulty than the original hardcoded version. As problem complexity increases (longer strings, more abstract relationships), adaptive metrics should maintain appropriateness while fixed constants become less suitable. We predict smaller variance in answer quality and convergence time for the graph-based system.

Hypothesis 2: Temperature-Graph Entropy Correlation System temperature should correlate with graph-theoretical measures of disorder. Specifically, we predict that temperature inversely correlates with Workspace graph clustering coefficient (high clustering = low temperature) and correlates with betweenness centrality variance (many objects with very different centralities = high temperature). This would validate temperature as reflecting structural coherence.

Hypothesis 3: Clustering Predicts Success Successful problem-solving runs should show systematically higher average clustering coefficients in their final Workspace graphs than failed or incomplete runs. This would support the hypothesis that locally dense structure promotes coherent analogies.

Hypothesis 4: Betweenness Predicts Correspondence Selection Objects with higher time-averaged betweenness centrality should be preferentially selected for correspondences. Plotting correspondence formation time against prior betweenness should show positive correlation, demonstrating that strategic structural position determines mapping priority.

Hypothesis 5: Graceful Degradation When problem difficulty increases (e.g., moving from 3-letter to 10-letter strings), graph-based Copycat should show more graceful performance degradation than the hardcoded version. We predict a smooth decline in success rate rather than a sharp cliff, since metrics scale continuously.

These hypotheses can be tested by implementing the graph-based modifications and running benchmark comparisons. The original Copycat's behavior is well-documented, providing a baseline for comparison. Running both versions on extended problem sets (varying string length, transformation complexity, and domain characteristics) would generate the data needed to evaluate these predictions.

5.5 Connections to Related Work

The graph-theoretical reformulation of Copycat connects to several research streams in cognitive science, artificial intelligence, and neuroscience.

Analogical Reasoning Structure-mapping theory [5] emphasizes systematic structural alignment in analogy-making. Gentner's approach explicitly compares relational structures, seeking one-to-one correspondences that preserve higher-order relationships. Our graph formulation makes this structuralism more precise: analogies correspond to graph homomorphisms that preserve edge labels and maximize betweenness-weighted node matches. The resistance distance formulation of slippage provides a quantitative measure of "systematicity"—slippages along short resistance paths maintain more structural similarity than jumps across large distances.

Graph Neural Networks Modern graph neural networks (GNNs) [10] learn to compute node and edge features through message passing on graphs. The Copycat reformulation suggests a potential hybrid: use GNNs to learn graph metric computations from data rather than relying on fixed formulas like betweenness. The GNN could learn to predict which objects deserve high salience based on training examples, potentially discovering novel structural patterns that standard metrics miss. Conversely, Copycat's symbolic structure could provide interpretability to GNN analogical reasoning systems.

Conceptual Spaces Gärdenfors' conceptual spaces framework [4] represents concepts geometrically, with similarity as distance in a metric space. The resistance distance reformulation of the Slipnet naturally produces a metric space: resistance distance satisfies the triangle inequality and provides a true distance measure over concepts. This connects Copycat to the broader conceptual spaces program and suggests using dimensional reduction techniques to visualize the conceptual geometry.

Small-World Networks Neuroscience research reveals that brain networks exhibit small-world properties: high local clustering combined with short path lengths between distant regions [11]. The Slipnet's structure shows similar characteristics—abstract concepts cluster together (high local clustering) while remaining accessible from concrete concepts (short paths). This parallel suggests that graph properties successful in natural cognitive architectures may also benefit artificial systems.

Network Science in Cognition Growing research applies network science methods to cognitive phenomena: semantic networks, problem-solving processes, and knowledge representation [9]. The Copycat reformulation contributes to this trend by demonstrating that a symbolic cognitive architecture can be rigorously analyzed through graph-theoretical lenses. The approach may generalize to other cognitive architectures, suggesting a broader research program of graph-based cognitive modeling.

5.6 Limitations and Open Questions

Despite its advantages, the graph-theoretical reformulation faces challenges and raises open questions.

Parameter Selection While graph metrics eliminate many hardcoded constants, some parameters remain. The resistance distance formulation requires choosing α (the decay parameter in $\exp(-\alpha R_{ij})$). The conceptual depth scaling requires selecting k. The betweenness normalization could use different schemes (min-max, z-score, etc.). These choices have less impact than the original hardcoded constants and can be derived more principally (e.g., α from temperature), but complete parameter elimination remains elusive.

Multi-Relational Graphs The Slipnet contains multiple edge types (category, instance, property, slip, non-slip links). Standard graph metrics like betweenness treat all edges identically. Properly handling multi-relational graphs requires either edge-type-specific metrics or careful encoding of edge types into weights. Research on knowledge graph embeddings may offer solutions.

Temporal Dynamics The Workspace graph evolves over time, but graph metrics provide static snapshots. Capturing temporal patterns—how centrality changes, whether oscillations occur, what trajectory successful runs follow—requires time-series analysis of graph metrics. Dynamic graph theory and temporal network analysis offer relevant techniques but have not yet been integrated into the Copycat context.

Learning and Meta-Learning The current proposal manually specifies which graph metric replaces which constant (betweenness for salience, clustering for support, etc.). Could the system learn these associations from experience? Meta-learning approaches might discover that different graph metrics work best for different problem types, automatically adapting the metric selection strategy.

5.7 Broader Implications

Beyond Copycat specifically, this work demonstrates a general methodology for modernizing legacy AI systems. Many symbolic AI systems from the 1980s and 1990s contain hardcoded parameters tuned for specific domains. Graph-theoretical reformulation offers a pathway to increase their adaptability and theoretical grounding. The approach represents a middle ground between purely symbolic AI (which risks brittleness through excessive hardcoding) and purely statistical AI (which risks opacity through learned parameters). Graph metrics provide structure while remaining adaptive.

The reformulation also suggests bridges between symbolic and neural approaches. Graph neural networks could learn to compute custom metrics for specific domains while maintaining interpretability through graph visualization. Copycat's symbolic constraints (objects, bonds, correspondences) could provide inductive biases for neural analogy systems. This hybrid direction may prove more fruitful than purely symbolic or purely neural approaches in isolation.

6 Conclusion

This paper has proposed a comprehensive graph-theoretical reformulation of the Copycat architecture. We identified numerous hardcoded constants in the original implementation—including bond compatibility factors, support decay functions, salience weights, and activation thresholds—that lack principled justification and limit adaptability. For each constant, we proposed a graph metric replacement: structural equivalence for compatibility, clustering coefficients for local support, betweenness centrality for salience, resistance distance for slippage, and percolation thresholds for activation.

These replacements provide three key advantages. Theoretically, they rest on established mathematical frameworks with proven properties and extensive prior research. Practically, they adapt automatically to problem structure without requiring manual retuning for new domains. Cognitively, they align with modern understanding of brain networks and relational cognition.

The reformulation reinterprets both major components of Copycat's architecture. The Slipnet becomes a weighted graph where conceptual depth emerges from minimum distance to concrete nodes and slippage derives from

resistance distance between concepts. The Workspace becomes a dynamic graph where object salience reflects betweenness centrality and structural support derives from clustering coefficients. Standard graph algorithms can compute these metrics efficiently for Copycat's typical graph sizes.

6.1 Future Work

Several directions promise to extend and validate this work:

Implementation and Validation The highest priority is building a prototype graph-based Copycat and empirically testing the hypotheses proposed in Section 5.3. Comparing performance between original and graph-based versions on extended problem sets would quantify the benefits of adaptability. Analyzing correlation between graph metrics and behavioral outcomes (correspondence selection, answer quality) would validate the theoretical predictions.

Domain Transfer Testing graph-based Copycat on non-letter-string domains (visual analogies, numerical relationships, abstract concepts) would demonstrate genuine adaptability. The original hardcoded constants would require complete retuning for such domains, while graph metrics should transfer directly. Success in novel domains would provide strong evidence for the reformulation's value.

Neuroscience Comparison Comparing Copycat's graph metrics to brain imaging data during human analogy-making could test cognitive plausibility. Do brain regions with high betweenness centrality show increased activation during analogy tasks? Does clustering in functional connectivity correlate with successful analogy completion? Such comparisons would ground the computational model in neural reality.

Hybrid Neural-Symbolic Systems Integrating graph neural networks to learn custom metrics for specific problem types represents an exciting direction. Rather than manually specifying betweenness for salience, a GNN could learn which graph features predict important objects, potentially discovering novel structural patterns. This would combine symbolic interpretability with neural adaptability.

Meta-Learning Metric Selection Developing meta-learning systems that automatically discover which graph metrics work best for which problem characteristics would eliminate remaining parameter choices. The system could learn from experience that betweenness centrality predicts importance for spatial problems while eigenvector centrality works better for temporal problems, adapting its metric selection strategy.

Extension to Other Cognitive Architectures The methodology developed here—identifying hardcoded constants and replacing them with graph metrics—may apply to other symbolic cognitive architectures. Systems like SOAR, ACT-R, and Companion [2] similarly contain numerous parameters that could potentially be reformulated graph-theoretically. This suggests a broader research program of graph-based cognitive architecture design.

6.2 Closing Perspective

The hardcoded constants in Copycat's original implementation represented practical necessities given the computational constraints and theoretical understanding of the early 1990s. Mitchell and Hofstadter made pragmatic choices that enabled the system to work, demonstrating fluid analogical reasoning for the first time in a computational model. These achievements deserve recognition.

Three decades later, we can build on this foundation with tools unavailable to the original designers. Graph theory has matured into a powerful analytical framework. Computational resources enable real-time calculation of complex metrics. Understanding of cognitive neuroscience has deepened, revealing the brain's graph-like organization. Modern machine learning offers hybrid symbolic-neural approaches. These advances create opportunities to refine Copycat's architecture while preserving its core insights about fluid cognition.

The graph-theoretical reformulation honors Copycat's original vision—modeling analogy-making as parallel constraint satisfaction over structured representations—while addressing its limitations. By replacing hardcoded heuristics with principled constructs, we move toward cognitive architectures that are both theoretically grounded and practically adaptive. This represents not a rejection of symbolic AI but rather its evolution, incorporating modern graph theory and network science to build more robust and flexible cognitive models.

References

[1] Ulrik Brandes. A faster algorithm for betweenness centrality. *Journal of Mathematical Sociology*, 25(2):163–177, 2001.

[2] Kenneth D. Forbus and Thomas R. Hinrichs. Companion cognitive systems: A step toward human-level ai. *AI Magazine*, 38(4):25–35, 2017.

[3] Linton C. Freeman. A set of measures of centrality based on betweenness. *Sociometry*, 40(1):35–41, 1977.

[4] Peter Gärdenfors. *Conceptual Spaces: The Geometry of Thought*. MIT Press, Cambridge, MA, 2000.

[5] Dedre Gentner. Structure-mapping: A theoretical framework for analogy. *Cognitive Science*, 7(2):155–170, 1983.

[6] Douglas R. Hofstadter and FARG. *Fluid Concepts and Creative Analogies: Computer Models of the Fundamental Mechanisms of Thought*. Basic Books, New York, NY, 1995.

[7] Douglas J. Klein and Milan Randić. Resistance distance. *Journal of Mathematical Chemistry*, 12(1):81–95, 1993.

[8] Melanie Mitchell. *Analogy-Making as Perception: A Computer Model*. MIT Press, Cambridge, MA, 1993.

[9] Mark E. J. Newman. *Networks*. Oxford University Press, Oxford, UK, 2nd edition, 2018.

[10] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2008.

[11] Duncan J. Watts and Steven H. Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 393(6684):440–442, 1998.