وحدة تفسير الأحلام في نظام بصيرة

نظرة عامة

وحدة تفسير الأحلام هي مكون متقدم في نظام بصيرة يهدف إلى تحليل وتفسير الأحلام باستخدام مزيج من التحليل الرمزي، والتحليل الدلالي، والتعرف على الأنماط، والتكامل مع نظام الخبير/المستكشف. تعتمد هذه الوحدة على المعادلات الرياضية التكيفية بدلاً من الشبكات العصبية التقليدية، مما يوفر نهجًا مبتكرًا وفريدًا لفهم وتفسير الأحلام.

الهيكل العام

تتكون وحدة تفسير الأحلام من المكونات الرئيسية التالية:

1. **المفسر الرئيسي (DreamInterpreter)**: المكون المركزي الذي ينسق بين جميع المكونات الأخرى ويوفر واجهة موحدة للتفاعل مع الوحدة.
2. **المحلل الرمزي (SymbolicAnalyzer)**: مسؤول عن تحليل الرموز والإشارات في الأحلام وربطها بمعانيها العميقة.
3. **المحلل الدلالي (SemanticAnalyzer)**: يحلل المعاني والدلالات في الأحلام ويربطها بالسياقات المختلفة.
4. **نظام التعرف على الأنماط (PatternRecognizer)**: يكتشف الأنماط المتكررة والعلاقات في الأحلام.
5. **تكامل الخبير/المستكشف (ExpertExplorerIntegration)**: يربط وحدة تفسير الأحلام بنظام الخبير والمستكشف التطوري.
6. **محرك التكامل (IntegrationEngine)**: يدمج نتائج التحليلات المختلفة لتقديم تفسير شامل ومتكامل.

المكونات التفصيلية

المفسر الرئيسي (DreamInterpreter)

المفسر الرئيسي هو نقطة الدخول الرئيسية لوحدة تفسير الأحلام. يقوم بتنسيق العمل بين المكونات المختلفة ويوفر واجهة برمجة بسيطة للمستخدمين.

```
class DreamInterpreter:
  def __init__(self):
    self.symbolic_analyzer = SymbolicAnalyzer()
    self.semantic_analyzer = SemanticAnalyzer()
    self.pattern_recognizer = PatternRecognizer()
    self.expert_explorer_integration = ExpertExplorerIntegration()
    self.integration_engine = IntegrationEngine()
```

```
    def interpret_dream(self, dream_text, context=None):
        """
        تفسير حلم بناءً على النص المدخل والسياق.

        Args:
            dream_text: نص الحلم
            context: سياق الحلم (اختياري)

        Returns:
            تفسير الحلم
        """
        # تحليل رمزي
        symbolic_results = self.symbolic_analyzer.analyze(dream_text, context)

        # تحليل دلالي
        semantic_results = self.semantic_analyzer.analyze(dream_text,
symbolic_results, context)

        # التعرف على الأنماط
        pattern_results = self.pattern_recognizer.recognize_patterns(dream_text,
symbolic_results, semantic_results)

        # تكامل مع نظام الخبير/المستكشف
        expert_results = self.expert_explorer_integration.integrate(
            dream_text, symbolic_results, semantic_results, pattern_results, context
        )

        # دمج النتائج
        interpretation = self.integration_engine.integrate_results(
            dream_text, symbolic_results, semantic_results, pattern_results,
expert_results, context
        )

        return interpretation
```

المحلل الرمزي (SymbolicAnalyzer)

المحلل الرمزي مسؤول عن تحليل الرموز والإشارات في الأحلام وربطها بمعانيها العميقة. يستخدم قاعدة بيانات من الرموز والمعاني، ويطبق خوارزميات متقدمة لاستخراج الرموز من نص الحلم وتحليلها.

```
class SymbolicAnalyzer:
    def __init__(self):
        self.symbol_database = self._load_symbol_database()
        self.symbolic_engine = AdvancedSymbolicExpression("x")

    def _load_symbol_database(self):
        """
```

```
        تحميل قاعدة بيانات الرموز.

        Returns:
            قاعدة بيانات الرموز
        """
        # تحميل قاعدة البيانات من ملف أو مصدر آخر
        return {}

    def analyze(self, dream_text, context=None):
        """
        تحليل رمزي للحلم.

        Args:
            dream_text: نص الحلم
            context: سياق الحلم (اختياري)

        Returns:
            نتائج التحليل الرمزي
        """
        # استخراج الرموز من النص
        symbols = self._extract_symbols(dream_text)

        # تحليل الرموز
        symbolic_analysis = self._analyze_symbols(symbols, context)

        # تطبيق المعادلات الرمزية
        symbolic_equations = self._apply_symbolic_equations(symbols,
symbolic_analysis, context)

        return {
            "symbols": symbols,
            "symbolic_analysis": symbolic_analysis,
            "symbolic_equations": symbolic_equations
        }

    def _extract_symbols(self, dream_text):
        """
        استخراج الرموز من نص الحلم.

        Args:
            dream_text: نص الحلم

        Returns:
            قائمة الرموز المستخرجة
        """
        # تقسيم النص إلى كلمات
        words = dream_text.split()

        # استخراج الرموز المعروفة
        symbols = []
```

```
        for word in words:
            if word.lower() in self.symbol_database:
                symbols.append({
                    "word": word,
                    "symbol": self.symbol_database[word.lower()],
                    "position": dream_text.find(word)
                })

        return symbols

    def _analyze_symbols(self, symbols, context=None):
        """
        تحليل الرموز المستخرجة.

        Args:
            symbols: قائمة الرموز
            context: سياق الحلم (اختياري)

        Returns:
            تحليل الرموز
        """
        # تحليل كل رمز
        analysis = []
        for symbol in symbols:
            symbol_info = self.symbol_database.get(symbol["word"].lower(), {})

            # تحليل الرمز بناءً على السياق
            if context:
                # تعديل التحليل بناءً على السياق
                pass

            analysis.append({
                "symbol": symbol["word"],
                "meaning": symbol_info.get("meaning", ""),
                "associations": symbol_info.get("associations", []),
                "emotional_impact": symbol_info.get("emotional_impact", "")
            })

        return analysis

    def _apply_symbolic_equations(self, symbols, symbolic_analysis, context=None):
        """
        تطبيق المعادلات الرمزية على الرموز المحللة.

        Args:
            symbols: قائمة الرموز
            symbolic_analysis: تحليل الرموز
            context: سياق الحلم (اختياري)

        Returns:
```

```
            المعادلات الرمزية
        """
        # إنشاء معادلات رمزية لتمثيل العلاقات بين الرموز
        equations = []

        # مثال بسيط: إنشاء معادلة لكل رمز
        for i, symbol in enumerate(symbols):
            try:
                # إنشاء تعبير رمزي بسيط
                expr_str = f"symbol_{i} * intensity_{i}"
                equation = AdvancedSymbolicExpression(expr_str)

                # إضافة معلومات وصفية
                equation.metadata.name = f"Symbol Equation {i}"
                equation.metadata.description = f"Equation for symbol: {symbol['word']}"

                equations.append(equation)
            except Exception as e:
                print(f"Error creating symbolic equation: {e}")

        return equations
```

المحلل الدلالي (SemanticAnalyzer)

المحلل الدلالي يحلل المعاني والدلالات في الأحلام ويربطها بالسياقات المختلفة. يستخدم تقنيات تحليل اللغة الطبيعية والتكامل الدلالي لفهم المعاني العميقة للأحلام.

```
class SemanticAnalyzer:
    def __init__(self):
        self.semantic_integration = SemanticIntegration()

    def analyze(self, dream_text, symbolic_results=None, context=None):
        """
        تحليل دلالي للحلم.

        Args:
            dream_text: نص الحلم
            symbolic_results: نتائج التحليل الرمزي (اختياري)
            context: سياق الحلم (اختياري)

        Returns:
            نتائج التحليل الدلالي
        """
        # استخراج المفاهيم الدلالية
        concepts = self._extract_concepts(dream_text)

        # تحليل العلاقات بين المفاهيم
```

```
    relations = self._analyze_relations(concepts, symbolic_results, context)

    # إنشاء شبكة دلالية
    semantic_network = self._create_semantic_network(concepts, relations)

    # تحليل السياق الدلالي
    context_analysis = self._analyze_context(semantic_network, context)

    return {
        "concepts": concepts,
        "relations": relations,
        "semantic_network": semantic_network,
        "context_analysis": context_analysis
    }

def _extract_concepts(self, dream_text):
    """
    استخراج المفاهيم الدلالية من نص الحلم.

    Args:
        dream_text: نص الحلم

    Returns:
        قائمة المفاهيم المستخرجة
    """
    # تقسيم النص إلى جمل
    sentences = dream_text.split('.')

    # استخراج المفاهيم من كل جملة
    concepts = []
    for sentence in sentences:
        if not sentence.strip():
            continue

        # استخراج المفاهيم الرئيسية
        words = sentence.strip().split()
        for word in words:
            if len(word) > 3:  # تجاهل الكلمات القصيرة
                concepts.append({
                    "word": word,
                    "position": dream_text.find(word),
                    "sentence": sentence.strip()
                })

    return concepts

def _analyze_relations(self, concepts, symbolic_results=None, context=None):
    """
    تحليل العلاقات بين المفاهيم.
```

```
        Args:
            concepts: قائمة المفاهيم
            symbolic_results: (اختياري) نتائج التحليل الرمزي
            context: (اختياري) سياق الحلم

        Returns:
            العلاقات بين المفاهيم
        """
        relations = []

        # تحليل العلاقات بين المفاهيم
        for i, concept1 in enumerate(concepts):
            for j, concept2 in enumerate(concepts[i+1:], i+1):
                # تحقق من وجود علاقة بين المفهومين
                if concept1["sentence"] == concept2["sentence"]:
                    # المفهومان في نفس الجملة
                    relation_type = "same_sentence"
                    strength = 0.8
                else:
                    # المفهومان في جمل مختلفة
                    relation_type = "different_sentence"
                    strength = 0.3

                relations.append({
                    "concept1": concept1["word"],
                    "concept2": concept2["word"],
                    "type": relation_type,
                    "strength": strength
                })

        return relations

    def _create_semantic_network(self, concepts, relations):
        """
        إنشاء شبكة دلالية من المفاهيم والعلاقات.

        Args:
            concepts: قائمة المفاهيم
            relations: العلاقات بين المفاهيم

        Returns:
            الشبكة الدلالية
        """
        # إنشاء شبكة دلالية بسيطة
        network = {
            "nodes": [],
            "edges": []
        }

        # إضافة المفاهيم كعقد
```

```
        for concept in concepts:
            network["nodes"].append({
                "id": concept["word"],
                "label": concept["word"],
                "type": "concept"
            })

        # إضافة العلاقات كحواف
        for relation in relations:
            network["edges"].append({
                "source": relation["concept1"],
                "target": relation["concept2"],
                "type": relation["type"],
                "weight": relation["strength"]
            })

        return network

    def _analyze_context(self, semantic_network, context=None):
        """
        تحليل السياق الدلالي.

        Args:
            semantic_network: الشبكة الدلالية
            context: سياق الحلم (اختياري)

        Returns:
            تحليل السياق
        """
        # تحليل السياق بناءً على الشبكة الدلالية
        context_analysis = {
            "main_themes": [],
            "emotional_tone": "",
            "temporal_setting": "",
            "spatial_setting": ""
        }

        # استخراج المواضيع الرئيسية
        if semantic_network["nodes"]:
            # حساب درجة كل عقدة (عدد الحواف المتصلة بها)
            node_degrees = {}
            for edge in semantic_network["edges"]:
                source = edge["source"]
                target = edge["target"]
                node_degrees[source] = node_degrees.get(source, 0) + 1
                node_degrees[target] = node_degrees.get(target, 0) + 1

            # اختيار العقد ذات أعلى درجة كمواضيع رئيسية
            sorted_nodes = sorted(node_degrees.items(), key=lambda x: x[1],
reverse=True)
            context_analysis["main_themes"] = [node for node, degree in sorted_nodes[:
```

```
3]]

        return context_analysis
```

نظام التعرف على الأنماط (PatternRecognizer)

نظام التعرف على الأنماط يكتشف الأنماط المتكررة والعلاقات في الأحلام. يستخدم خوارزميات متقدمة للتعرف على الأنماط الزمنية والمكانية والرمزية.

```
class PatternRecognizer:
    def __init__(self):
        self.pattern_database = self._load_pattern_database()

    def _load_pattern_database(self):
        """
        تحميل قاعدة بيانات الأنماط.

        Returns:
            قاعدة بيانات الأنماط
        """
        # تحميل قاعدة البيانات من ملف أو مصدر آخر
        return {
            "temporal": [],
            "spatial": [],
            "symbolic": [],
            "narrative": []
        }

    def recognize_patterns(self, dream_text, symbolic_results=None,
semantic_results=None):
        """
        التعرف على الأنماط في الحلم.

        Args:
            dream_text: نص الحلم
            symbolic_results: نتائج التحليل الرمزي (اختياري)
            semantic_results: نتائج التحليل الدلالي (اختياري)

        Returns:
            الأنماط المكتشفة
        """
        # التعرف على الأنماط الزمنية
        temporal_patterns = self._recognize_temporal_patterns(dream_text,
symbolic_results, semantic_results)

        # التعرف على الأنماط المكانية
        spatial_patterns = self._recognize_spatial_patterns(dream_text,
symbolic_results, semantic_results)
```

```
        # التعرف على الأنماط الرمزية
        symbolic_patterns = self._recognize_symbolic_patterns(dream_text,
symbolic_results, semantic_results)

        # التعرف على الأنماط السردية
        narrative_patterns = self._recognize_narrative_patterns(dream_text,
symbolic_results, semantic_results)

        return {
            "temporal_patterns": temporal_patterns,
            "spatial_patterns": spatial_patterns,
            "symbolic_patterns": symbolic_patterns,
            "narrative_patterns": narrative_patterns
        }

    def _recognize_temporal_patterns(self, dream_text, symbolic_results=None,
semantic_results=None):
        """
        التعرف على الأنماط الزمنية.

        Args:
            dream_text: نص الحلم
            symbolic_results: نتائج التحليل الرمزي (اختياري)
            semantic_results: نتائج التحليل الدلالي (اختياري)

        Returns:
            الأنماط الزمنية المكتشفة
        """
        # البحث عن كلمات تدل على الزمن
        temporal_keywords = ["قبل", "بعد", "أثناء", "خلال", "اليوم", "الأمس", "غدًا", "الآن", "سابقًا",
"لاحقًا"]

        patterns = []
        for keyword in temporal_keywords:
            if keyword in dream_text.lower():
                patterns.append({
                    "type": "temporal",
                    "keyword": keyword,
                    "position": dream_text.lower().find(keyword)
                })

        return patterns

    def _recognize_spatial_patterns(self, dream_text, symbolic_results=None,
semantic_results=None):
        """
        التعرف على الأنماط المكانية.

        Args:
```

```
        dream_text: نص الحلم
        symbolic_results: (اختياري) نتائج التحليل الرمزي
        semantic_results: (اختياري) نتائج التحليل الدلالي

    Returns:
        الأنماط المكانية المكتشفة
    """
    # البحث عن كلمات تدل على المكان
    spatial_keywords = ["فوق", "تحت", "يمين", "يسار", "أمام", "خلف", "داخل", "خارج",
"بجانب", "بين"]

    patterns = []
    for keyword in spatial_keywords:
        if keyword in dream_text.lower():
            patterns.append({
                "type": "spatial",
                "keyword": keyword,
                "position": dream_text.lower().find(keyword)
            })

    return patterns

def _recognize_symbolic_patterns(self, dream_text, symbolic_results=None,
semantic_results=None):
    """
    التعرف على الأنماط الرمزية.

    Args:
        dream_text: نص الحلم
        symbolic_results: (اختياري) نتائج التحليل الرمزي
        semantic_results: (اختياري) نتائج التحليل الدلالي

    Returns:
        الأنماط الرمزية المكتشفة
    """
    patterns = []

    # استخدام نتائج التحليل الرمزي إذا كانت متاحة
    if symbolic_results and "symbols" in symbolic_results:
        symbols = symbolic_results["symbols"]

        # البحث عن أنماط تكرار الرموز
        symbol_counts = {}
        for symbol in symbols:
            symbol_word = symbol["word"].lower()
            symbol_counts[symbol_word] = symbol_counts.get(symbol_word, 0) + 1

        # إضافة الرموز المتكررة كأنماط
        for symbol_word, count in symbol_counts.items():
            if count > 1:
```

```
                patterns.append({
                    "type": "symbolic_repetition",
                    "symbol": symbol_word,
                    "count": count
                })

        return patterns

    def _recognize_narrative_patterns(self, dream_text, symbolic_results=None,
semantic_results=None):
        """
        التعرف على الأنماط السردية.

        Args:
            dream_text: نص الحلم
            symbolic_results: نتائج التحليل الرمزي (اختياري)
            semantic_results: نتائج التحليل الدلالي (اختياري)

        Returns:
            الأنماط السردية المكتشفة
        """
        # البحث عن أنماط سردية معروفة
        narrative_patterns = [
            {"name": "المطاردة", "keywords": ["مطاردة", "هروب", "ملاحقة", "يلاحق", "يطارد",
"يهرب"]},
            {"name": "السقوط", "keywords": ["سقوط", "يسقط", "سقط", "يقع", "وقع"]},
            {"name": "الطيران", "keywords": ["طيران", "يطير", "طار", "تحليق", "يحلق", "حلق"]},
            {"name": "الاختباء", "keywords": ["اختباء", "يختبئ", "اختبأ", "مخبأ", "يختفي", "اختفى"]},
            {"name": "البحث", "keywords": ["بحث", "يبحث", "بحث عن", "يفتش", "فتش"]}
        ]

        patterns = []
        for pattern in narrative_patterns:
            for keyword in pattern["keywords"]:
                if keyword in dream_text.lower():
                    patterns.append({
                        "type": "narrative",
                        "pattern_name": pattern["name"],
                        "keyword": keyword,
                        "position": dream_text.lower().find(keyword)
                    })
                    break  # اكتشاف كلمة واحدة يكفي لتحديد النمط

        return patterns
```

تكامل الخبير/المستكشف (ExpertExplorerIntegration)

تكامل الخبير/المستكشف يربط وحدة تفسير الأحلام بنظام الخبير والمستكشف التطوري. يستخدم نظام الخبير لتطبيق قواعد التفسير، والمستكشف التطوري لاكتشاف أنماط وعلاقات جديدة.

```
class ExpertExplorerIntegration:
    def __init__(self):
        self.expert_system = AdvancedExpertSystem()
        self.evolutionary_explorer = EvolutionaryExplorer()

    def integrate(self, dream_text, symbolic_results, semantic_results,
pattern_results, context=None):
        """
        تكامل مع نظام الخبير/المستكشف.

        Args:
            dream_text: نص الحلم
            symbolic_results: نتائج التحليل الرمزي
            semantic_results: نتائج التحليل الدلالي
            pattern_results: نتائج التعرف على الأنماط
            context: سياق الحلم (اختياري)

        Returns:
            نتائج التكامل
        """
        # تطبيق قواعد نظام الخبير
        expert_analysis = self._apply_expert_rules(dream_text, symbolic_results,
semantic_results, pattern_results, context)

        # استكشاف أنماط وعلاقات جديدة
        exploration_results = self._explore_patterns(dream_text, symbolic_results,
semantic_results, pattern_results, context)

        # دمج نتائج الخبير والمستكشف
        integrated_results = self._integrate_results(expert_analysis,
exploration_results)

        return integrated_results

    def _apply_expert_rules(self, dream_text, symbolic_results, semantic_results,
pattern_results, context=None):
        """
        تطبيق قواعد نظام الخبير.

        Args:
            dream_text: نص الحلم
            symbolic_results: نتائج التحليل الرمزي
            semantic_results: نتائج التحليل الدلالي
```

```
        pattern_results: نتائج التعرف على الأنماط
        context: (اختياري) سياق الحلم

    Returns:
        نتائج تطبيق قواعد الخبير
    """
    # إنشاء قاعدة معرفة من نتائج التحليل
    knowledge_base = {
        "dream_text": dream_text,
        "symbols": symbolic_results.get("symbols", []),
        "concepts": semantic_results.get("concepts", []),
        "relations": semantic_results.get("relations", []),
        "patterns": {
            "temporal": pattern_results.get("temporal_patterns", []),
            "spatial": pattern_results.get("spatial_patterns", []),
            "symbolic": pattern_results.get("symbolic_patterns", []),
            "narrative": pattern_results.get("narrative_patterns", [])
        }
    }

    # تطبيق قواعد الخبير
    expert_analysis = self.expert_system.apply_rules(knowledge_base)

    return expert_analysis

def _explore_patterns(self, dream_text, symbolic_results, semantic_results,
pattern_results, context=None):
    """
    استكشاف أنماط وعلاقات جديدة.

    Args:
        dream_text: نص الحلم
        symbolic_results: نتائج التحليل الرمزي
        semantic_results: نتائج التحليل الدلالي
        pattern_results: نتائج التعرف على الأنماط
        context: (اختياري) سياق الحلم

    Returns:
        نتائج الاستكشاف
    """
    # إنشاء معادلات رمزية من نتائج التحليل
    equations = []

    # إضافة معادلات من التحليل الرمزي
    if "symbolic_equations" in symbolic_results:
        equations.extend(symbolic_results["symbolic_equations"])

    # تكوين الاستكشاف
    config = ExplorationConfig(
        mode=ExplorationMode.GUIDED,
```

```
            budget=20,
            max_complexity=50.0,
            min_complexity=1.0,
            max_variables=5,
            min_variables=1,
            fitness_threshold=0.7
        )

        # استكشاف المعادلات
        exploration_result = self.evolutionary_explorer.explore(config,
seed_equations=equations)

        return {
            "discovered_equations": exploration_result.discovered_equations,
            "patterns_discovered": exploration_result.patterns_discovered,
            "statistics": exploration_result.statistics
        }

    def _integrate_results(self, expert_analysis, exploration_results):
        """
        دمج نتائج الخبير والمستكشف.

        Args:
            expert_analysis: نتائج تطبيق قواعد الخبير
            exploration_results: نتائج الاستكشاف

        Returns:
            النتائج المدمجة
        """
        # دمج النتائج
        integrated_results = {
            "expert_analysis": expert_analysis,
            "exploration_results": exploration_results,
            "integrated_insights": []
        }

        # استخراج الرؤى المتكاملة
        if "interpretations" in expert_analysis:
            for interpretation in expert_analysis["interpretations"]:
                integrated_results["integrated_insights"].append({
                    "source": "expert",
                    "insight": interpretation
                })

        if "patterns_discovered" in exploration_results:
            for pattern in exploration_results["patterns_discovered"]:
                integrated_results["integrated_insights"].append({
                    "source": "explorer",
                    "insight": {
                        "pattern_name": pattern.get("pattern_name", ""),
                        "description": pattern.get("description", "")
```

```
            }
        })

    return integrated_results
```

محرك التكامل (IntegrationEngine)

محرك التكامل يدمج نتائج التحليلات المختلفة لتقديم تفسير شامل ومتكامل للحلم.

```
class IntegrationEngine:
    def __init__(self):
        pass

    def integrate_results(self, dream_text, symbolic_results, semantic_results,
pattern_results, expert_results, context=None):
        """
        دمج نتائج التحليلات المختلفة.

        Args:
            dream_text: نص الحلم
            symbolic_results: نتائج التحليل الرمزي
            semantic_results: نتائج التحليل الدلالي
            pattern_results: نتائج التعرف على الأنماط
            expert_results: نتائج تكامل الخبير/المستكشف
            context: سياق الحلم (اختياري)

        Returns:
            التفسير المتكامل
        """
        # استخراج العناصر الرئيسية من كل تحليل
        main_symbols = self._extract_main_symbols(symbolic_results)
        main_concepts = self._extract_main_concepts(semantic_results)
        main_patterns = self._extract_main_patterns(pattern_results)
        main_insights = self._extract_main_insights(expert_results)

        # دمج العناصر الرئيسية
        integrated_elements = self._merge_elements(main_symbols, main_concepts,
main_patterns, main_insights)

        # إنشاء تفسير متكامل
        interpretation = self._create_interpretation(dream_text, integrated_elements,
context)

        return interpretation

    def _extract_main_symbols(self, symbolic_results):
        """
        استخراج الرموز الرئيسية.
```

```
        Args:
            symbolic_results: نتائج التحليل الرمزي

        Returns:
            الرموز الرئيسية
        """
        main_symbols = []

        if "symbols" in symbolic_results:
            symbols = symbolic_results["symbols"]

            # اختيار الرموز الأكثر أهمية
            if "symbolic_analysis" in symbolic_results:
                analysis = symbolic_results["symbolic_analysis"]

                # دمج الرموز مع تحليلها
                for symbol in symbols:
                    for analysis_item in analysis:
                        if symbol["word"] == analysis_item["symbol"]:
                            main_symbols.append({
                                "symbol": symbol["word"],
                                "meaning": analysis_item.get("meaning", ""),
                                "associations": analysis_item.get("associations", []),
                                "emotional_impact": analysis_item.get("emotional_impact", "")
                            })
                            break

        return main_symbols

    def _extract_main_concepts(self, semantic_results):
        """
        استخراج المفاهيم الرئيسية.

        Args:
            semantic_results: نتائج التحليل الدلالي

        Returns:
            المفاهيم الرئيسية
        """
        main_concepts = []

        if "context_analysis" in semantic_results and "main_themes" in
semantic_results["context_analysis"]:
            main_themes = semantic_results["context_analysis"]["main_themes"]

            # استخراج المفاهيم المرتبطة بالمواضيع الرئيسية
            if "concepts" in semantic_results:
                concepts = semantic_results["concepts"]

                for theme in main_themes:
```

```
            for concept in concepts:
                if concept["word"] == theme:
                    main_concepts.append({
                        "concept": concept["word"],
                        "sentence": concept.get("sentence", "")
                    })
                    break

    return main_concepts

def _extract_main_patterns(self, pattern_results):
    """
    استخراج الأنماط الرئيسية.

    Args:
        pattern_results: نتائج التعرف على الأنماط

    Returns:
        الأنماط الرئيسية
    """
    main_patterns = []

    # استخراج الأنماط السردية
    if "narrative_patterns" in pattern_results:
        narrative_patterns = pattern_results["narrative_patterns"]

        for pattern in narrative_patterns:
            main_patterns.append({
                "type": "narrative",
                "name": pattern.get("pattern_name", ""),
                "keyword": pattern.get("keyword", "")
            })

    # استخراج الأنماط الرمزية المتكررة
    if "symbolic_patterns" in pattern_results:
        symbolic_patterns = pattern_results["symbolic_patterns"]

        for pattern in symbolic_patterns:
            if pattern.get("type") == "symbolic_repetition":
                main_patterns.append({
                    "type": "symbolic_repetition",
                    "symbol": pattern.get("symbol", ""),
                    "count": pattern.get("count", 0)
                })

    return main_patterns

def _extract_main_insights(self, expert_results):
    """
    استخراج الرؤى الرئيسية.
```

```
        Args:
            expert_results: نتائج تكامل الخبير/المستكشف

        Returns:
            الرؤى الرئيسية
        """
        main_insights = []

        if "integrated_insights" in expert_results:
            insights = expert_results["integrated_insights"]

            for insight in insights:
                main_insights.append({
                    "source": insight.get("source", ""),
                    "insight": insight.get("insight", {})
                })

        return main_insights

    def _merge_elements(self, main_symbols, main_concepts, main_patterns,
main_insights):
        """
        دمج العناصر الرئيسية.

        Args:
            main_symbols: الرموز الرئيسية
            main_concepts: المفاهيم الرئيسية
            main_patterns: الأنماط الرئيسية
            main_insights: الرؤى الرئيسية

        Returns:
            العناصر المدمجة
        """
        # دمج العناصر المختلفة
        integrated_elements = {
            "symbols": main_symbols,
            "concepts": main_concepts,
            "patterns": main_patterns,
            "insights": main_insights
        }

        return integrated_elements

    def _create_interpretation(self, dream_text, integrated_elements,
context=None):
        """
        إنشاء تفسير متكامل.

        Args:
            dream_text: نص الحلم
```

```
        integrated_elements: العناصر المدمجة
        context: (اختياري) سياق الحلم

    Returns:
        التفسير المتكامل
    """
    # إنشاء تفسير متكامل
    interpretation = {
        "dream_text": dream_text,
        "summary": self._create_summary(integrated_elements),
        "detailed_analysis": {
            "symbols": integrated_elements["symbols"],
            "concepts": integrated_elements["concepts"],
            "patterns": integrated_elements["patterns"]
        },
        "insights": integrated_elements["insights"],
        "recommendations": self._create_recommendations(integrated_elements,
context)
    }

    return interpretation

def _create_summary(self, integrated_elements):
    """
    إنشاء ملخص للتفسير.

    Args:
        integrated_elements: العناصر المدمجة

    Returns:
        ملخص التفسير
    """
    # إنشاء ملخص بناءً على العناصر المدمجة
    summary = {
        "main_theme": "",
        "emotional_tone": "",
        "key_symbols": [],
        "key_patterns": []
    }

    # استخراج الموضوع الرئيسي
    if integrated_elements["concepts"]:
        summary["main_theme"] = integrated_elements["concepts"][0]["concept"]

    # استخراج النبرة العاطفية
    emotional_impacts = [symbol.get("emotional_impact", "") for symbol in
integrated_elements["symbols"] if "emotional_impact" in symbol]
    if emotional_impacts:
        # اختيار النبرة العاطفية الأكثر تكرارًا
        from collections import Counter
```

```
            emotion_counter = Counter(emotional_impacts)
            summary["emotional_tone"] = emotion_counter.most_common(1)[0][0] if
emotion_counter else ""

        # استخراج الرموز الرئيسية
        summary["key_symbols"] = [symbol["symbol"] for symbol in
integrated_elements["symbols"][:3]]

        # استخراج الأنماط الرئيسية
        summary["key_patterns"] = [pattern["name"] if "name" in pattern else
pattern["type"] for pattern in integrated_elements["patterns"][:3]]

        return summary

    def _create_recommendations(self, integrated_elements, context=None):
        """
        إنشاء توصيات بناءً على التفسير.

        Args:
            integrated_elements: العناصر المدمجة
            context: سياق الحلم (اختياري)

        Returns:
            التوصيات
        """
        # إنشاء توصيات بناءً على العناصر المدمجة
        recommendations = []

        # توصيات بناءً على الرموز
        for symbol in integrated_elements["symbols"]:
            if "meaning" in symbol and symbol["meaning"]:
                recommendations.append({
                    "type": "symbol",
                    "symbol": symbol["symbol"],
                    "recommendation": f"تأمل في معنى الرمز '{symbol['symbol']}' وعلاقته بحياتك."
                })

        # توصيات بناءً على الأنماط
        for pattern in integrated_elements["patterns"]:
            if pattern.get("type") == "narrative":
                recommendations.append({
                    "type": "pattern",
                    "pattern": pattern.get("name", ""),
                    "recommendation": f"فكر في كيفية ارتباط نمط '{pattern.get('name',
'')}' بتجاربك الحالية."
                })

        return recommendations
```

التكامل مع النواة الرياضية

وحدة تفسير الأحلام تتكامل بشكل وثيق مع النواة الرياضية لنظام بصيرة، وخاصة مع المكونات التالية:

1. **محرك المعالجة الرمزية (SymbolicEngine)**: يستخدم لإنشاء وتحليل التعبيرات الرمزية التي تمثل عناصر الحلم وعلاقاتها.
2. **نظام الخبير (ExpertSystem)**: يطبق قواعد المعرفة لتفسير الأحلام بناءً على التحليلات المختلفة.
3. **المستكشف التطوري (EvolutionaryExplorer)**: يستكشف فضاء المعادلات والأنماط لاكتشاف علاقات وأنماط جديدة في الأحلام.
4. **التكامل الدلالي (SemanticIntegration)**: يربط المفاهيم والرموز في الأحلام بشبكة دلالية متكاملة.

الاستخدام

يمكن استخدام وحدة تفسير الأحلام كما يلي:

```
from dream_interpretation.core.dream_interpreter import DreamInterpreter

# إنشاء مفسر الأحلام
interpreter = DreamInterpreter()

# تفسير حلم
dream_text = "رأيت نفسي أطير فوق مدينة كبيرة، ثم سقطت فجأة في بحر عميق. كنت أسبح في البحر وأشعر بالخوف، ثم ظهر قارب وأنقذني."
interpretation = interpreter.interpret_dream(dream_text)

# عرض التفسير
print("ملخص التفسير:")
print(f"الموضوع الرئيسي: {interpretation['summary']['main_theme']}")
print(f"النبرة العاطفية: {interpretation['summary']['emotional_tone']}")
print(f"الرموز الرئيسية: {', '.join(interpretation['summary']['key_symbols'])}")
print(f"الأنماط الرئيسية: {', '.join(interpretation['summary']['key_patterns'])}")

print("\nالتوصيات:")
for recommendation in interpretation['recommendations']:
    print(f"- {recommendation['recommendation']}")
```

الخلاصة

وحدة تفسير الأحلام في نظام بصيرة توفر نهجًا متكاملًا ومبتكرًا لتحليل وتفسير الأحلام باستخدام المعادلات الرياضية التكيفية بدلاً من الشبكات العصبية التقليدية. تتكامل هذه الوحدة مع النواة الرياضية للنظام لتوفير تفسيرات عميقة ودقيقة للأحلام، مع اكتشاف الأنماط والعلاقات الخفية.