Анализ и улучшение системы промптов для извлечения исторических событий

Текущие сильные стороны системы

1. **Четкое разделение ролей**: Extractor и Verifier с разными задачами
2. **Структурированный JSON-выход** с валидацией через Pydantic
3. **Строгие критерии отбора** событий 1848-1849 гг.
4. **Система управления API** с rate limiting
5. **Обработка ошибок** и retry-механизмы

Проблемы текущих промптов

Проблема 1: Избыточная длина и сложность

- Промпт экстрактора очень длинный (>1000 токенов)
- Множественные инструкции могут конфликтовать
- Сложная структура может запутать модель

Проблема 2: Неоднозначности в критериях

- Граница между "прямо связанными" и "косвенно связанными" событиями нечеткая
- Недостаточно конкретных примеров

Проблема 3: Проблемы с контекстом

- `text_fragment` часто получается слишком коротким
- Недостаточно указаний по определению исторического контекста

Рекомендуемые улучшения

1. Улучшенный промпт для экстрактора

```python
EXTRACTOR_SYSTEM_PROMPT_V2 = """
Ты эксперт-историк по революциям 1848-1849 гг. в Европе. Анализируй дневниковые записи русскогс

СТРОГИЕ КРИТЕРИИ ВКЛЮЧЕНИЯ:
✓ Конкретные революционные события 1848-1849 (Февральская революция во Франции, Венские событи
✓ Российские реакции на эти события (военные меры, дипломатия, цензура)
✓ Обсуждения революций в российском обществе
✓ Личные размышления автора об этих событиях

ИСКЛЮЧЕНИЯ:
✗ События до 1848 или после 1849 года
✗ Внутрироссийские события без связи с европейскими революциями
✗ Общие политические рассуждения без привязки к 1848-1849
✗ Культурные/литературные упоминания без революционного контекста

ПРОВЕРОЧНЫЙ ВОПРОС: "Можно ли это упоминание напрямую связать с революциями 1848-1849 гг.?" Есл
"""
```

2. Улучшенный пользовательский промпт для экстрактора

python

```
def create_extraction_prompt(entry_id, text, date, knowledge_map):
    return f"""
Дневниковая запись от {date} (ID: {entry_id}):
"{text}"

Карта знаний:
{knowledge_map}

ЗАДАЧА: Найди ВСЕ упоминания революций 1848-1849 гг. Для каждого:

1. Определи event_id по Карте знаний (или OTHER_1848/null)
2. Извлеки ПОЛНОЕ предложение с контекстом для text_fragment
3. Опиши событие своими словами на основе текста
4. Определи источник информации для автора дневника

ФОРМАТ: JSON массив объектов. Если ничего не найдено - пустой массив [].

Пример структуры:
{{
  "entry_id": {entry_id},
  "event_id": "RUSS_REAC_CENS_1848",
  "event_name": "Цензурные ограничения",
  "description": "Автор упоминает запрет на обсуждение французских событий",
  "text_fragment": "Полное предложение из текста с достаточным контекстом",
  "information_source": "Разговор с учителем",
  "confidence": "High",
  "keywords": ["цензура", "запрет", "Франция"]
}}
"""
```

3. Более четкий промпт для верификатора

```python
VERIFIER_SYSTEM_PROMPT_V2 = """
Ты контролер качества исторических данных. Проверяй извлеченную информацию о революциях 1848-18

ЗАДАЧИ:
1. Соответствие event_id содержанию text_fragment
2. Полнота и точность description
3. Достаточность контекста в text_fragment
4. Правильность классификации источников информации
5. Адекватность уровней confidence

ПРИНЦИПЫ:
- Исправляй только очевидные ошибки
- Расширяй text_fragment при необходимости
- Будь консервативен в изменениях
"""
```

4. Система примеров для Few-Shot Learning

python

```
EXAMPLE_EXTRACTIONS = """
ПРИМЕРЫ ПРАВИЛЬНОГО ИЗВЛЕЧЕНИЯ:

Текст: "Слышал сегодня, что во Франции опять беспорядки. Говорят, король бежал."
Результат: {
  "event_id": "FR_FEB_REV_1848",
  "event_name": "Февральская революция во Франции",
  "description": "Упоминание о беспорядках во Франции и бегстве короля",
  "text_fragment": "Слышал сегодня, что во Франции опять беспорядки. Говорят, король бежал.",
  "information_source": "Слухи/разговоры"
}

Текст: "Учитель сказал, что нам нельзя обсуждать французские дела."
Результат: {
  "event_id": "RUSS_REAC_CENS_1848",
  "event_name": "Цензурные ограничения",
  "description": "Запрет на обсуждение французских революционных событий",
  "text_fragment": "Учитель сказал, что нам нельзя обсуждать французские дела.",
  "information_source": "Учитель"
}
"""
```

5. Улучшенная валидация полей

python

```python
def enhanced_field_validation(event_dict):
    """Расширенная валидация полей с исправлениями"""

    # Проверка text_fragment на достаточность контекста
    fragment = event_dict.get('text_fragment', '')
    if len(fragment.split()) < 5:
        event_dict['confidence'] = 'Low'
        logger.warning(f"Короткий text_fragment: {fragment}")

    # Валидация соответствия event_id и event_name
    event_id = event_dict.get('event_id')
    event_name = event_dict.get('event_name')

    # Проверка логической связности
    description = event_dict.get('description', '')
    if 'революц' in description.lower() and not event_id:
        logger.warning("Упоминание революции без event_id")

    return event_dict
```

6. Динамическая адаптация промптов

python

```python
def adaptive_prompt_strategy(entry_text, previous_results):
    """Адаптация промпта на основе предыдущих результатов"""

    # Если много false positives - усилить критерии
    if previous_results['false_positive_rate'] > 0.3:
        return "СТРОЖЕ: Включай только явные упоминания революций 1848-1849."

    # Если много пропусков - смягчить критерии
    if previous_results['recall'] < 0.7:
        return "ВНИМАТЕЛЬНЕЕ: Ищи косвенные упоминания и реакции на революции."

    return ""
```

7. Система проверки качества

python

```python
def quality_metrics(extracted_events, original_text):
    """Метрики качества извлечения"""

    metrics = {
        'events_count': len(extracted_events),
        'avg_fragment_length': np.mean([len(e['text_fragment'].split()) for e in extracted_ever
        'confidence_distribution': Counter([e['confidence'] for e in extracted_events]),
        'source_diversity': len(set([e['information_source'] for e in extracted_events])),
        'text_coverage': sum([len(e['text_fragment']) for e in extracted_events]) / len(origina
    }

    return metrics
```

Рекомендации по внедрению

Этап 1: Тестирование новых промптов

1. Протестировать на 50-100 записях
2. Сравнить качество с текущей системой
3. Измерить precision/recall вручную

Этап 2: A/B тестирование

1. Запустить параллельно старую и новую системы
2. Сравнить результаты на одинаковых данных
3. Выбрать лучшую конфигурацию

Этап 3: Итеративное улучшение

1. Анализировать ошибки классификации
2. Дополнять примеры в Few-Shot Learning
3. Настраивать пороги confidence

Дополнительные технические улучшения

1. Кэширование результатов

python

```python
import hashlib

def get_cache_key(text, prompt_version):
    return hashlib.md5((text + prompt_version).encode()).hexdigest()
```

2. Мониторинг качества в реальном времени

python

```python
def quality_monitor(event):
    if event['confidence'] == 'Low' and event['classification_confidence'] == 'Low':
        logger.warning(f"Сомнительное событие: {event['entry_id']}")
```

3. Автоматическая подстройка параметров

python

```python
def auto_tune_temperature(recent_results):
    if recent_results['consistency'] < 0.8:
        return 0.3  # Более детерминистично
    else:
        return 0.7  # Больше креативности
```

Эти улучшения должны значительно повысить точность извлечения событий при сохранении полноты данных.