Revisão Crítica - Implementação API Perplexity ExpertPlanner

🔍 Análise Geral

O projeto apresenta uma base sólida para integração com a API Perplexity, mas há várias oportunidades de melhoria baseadas nas melhores práticas observadas nos repositórios analisados e na experiência de implementação.

⚠️ Problemas Críticos Identificados

1. Ausência do `search_domain_filter`

Severidade: ALTA

O documento não menciona a implementação do filtro de domínios, que é uma funcionalidade essencial observada nos repositórios analisados:

```typescript
// ❌ Implementação atual (incompleta)
interface PerplexityConfig {
  apiKey: string;
  model: string;
  maxTokens: number;
  temperature: number;
}

// ✅ Implementação recomendada
interface PerplexityConfig {
  apiKey: string;
  model: string;
  maxTokens: number;
  temperature: number;
  searchDomainFilter?: string[];
  searchRecencyFilter?: 'hour' | 'day' | 'week' | 'month';
  webSearchOptions?: {
    searchContextSize: 'low' | 'medium' | 'high';
    userLocation?: {
      latitude: number;
      longitude: number;
      country: string;
    };
  };
}
```

2. Rate Limiting Inadequado

Severidade: ALTA

Os limites definidos (20 req/min) são muito conservadores e não implementam estratégias inteligentes:

```typescript
// ❌ Implementação atual
private maxRequestsPerMinute = 20;

// ✅ Implementação recomendada baseada em japelsin/pplx
class AdvancedRateLimiter {
  private config = {
    tier1: { requestsPerMinute: 20, requestsPerHour: 1000 },
    tier2: { requestsPerMinute: 60, requestsPerHour: 3000 },
    tier3: { requestsPerMinute: 100, requestsPerHour: 5000 }
  };

  private adaptiveBackoff = true;
  private burstCapacity = 5; // Permite rajadas controladas

  async makeRequest(request: PerplexityRequest): Promise<PerplexityResponse> {
    await this.waitForAvailableSlot();
    return this.executeWithRetry(request);
  }

  private async executeWithRetry(request: PerplexityRequest, attempt = 1): Promise<Per
    try {
      return await this.executeRequest(request);
    } catch (error) {
      if (this.isRateLimitError(error) && attempt < 3) {
        await this.exponentialBackoff(attempt);
        return this.executeWithRetry(request, attempt + 1);
      }
      throw error;
    }
  }
}
```

3. Falta de Validação de Fontes Robusta

Severidade: MÉDIA

Baseado no repositório `tom-doerr/perplexity_search`, é necessário implementar validação mais robusta:

typescript

```typescript
// ✅ Implementação recomendada
class SourceValidator {
  private trustedDomains = [
    'arxiv.org', 'nature.com', 'science.org', 'ieee.org',
    'pubmed.ncbi.nlm.nih.gov', 'scholar.google.com',
    'researchgate.net', 'wikipedia.org'
  ];

  private blacklistedDomains = [
    'pinterest.com', 'reddit.com', 'quora.com',
    'twitter.com', 'facebook.com'
  ];

  async validateSources(urls: string[]): Promise<SourceValidation[]> {
    return Promise.all(urls.map(async (url) => {
      const domain = this.extractDomain(url);
      const reliability = this.calculateReliability(domain);
      const accessibility = await this.checkAccessibility(url);
      const freshness = await this.checkContentFreshness(url);

      return {
        url,
        domain,
        reliability,
        accessibility,
        freshness,
        trusted: this.trustedDomains.includes(domain),
        blacklisted: this.blacklistedDomains.includes(domain)
      };
    }));
  }
}
```

🚀 Recomendações de Melhoria

1. Implementar Strategy Pattern para Modelos

typescript

```typescript
// Baseado no padrão observado em RossH121/perplexity-mcp
interface ModelStrategy {
  selectModel(query: string, context: ContentContext): string;
  configureParams(model: string): Partial<PerplexityConfig>;
}

class IntelligentModelSelector implements ModelStrategy {
  selectModel(query: string, context: ContentContext): string {
    const queryIntent = this.analyzeIntent(query);

    if (queryIntent.includes('research') || queryIntent.includes('comprehensive')) {
      return 'sonar-deep-research';
    }

    if (queryIntent.includes('reasoning') || queryIntent.includes('complex')) {
      return 'sonar-reasoning-pro';
    }

    if (context.contentType === 'fact_checking') {
      return 'sonar-reasoning';
    }

    return 'sonar-pro'; // Default
  }
}
```

2. Adicionar Contexto de Busca Inteligente

typescript

```typescript
// Inspirado em ppl-ai/api-cookbook
interface SearchContext {
  domainFilters: string[];
  recencyFilter?: string;
  locationContext?: LocationContext;
  expertiseArea: string;
  targetAudience: string;
}

class ContextualSearchBuilder {
  buildSearchParams(expertProfile: ExpertProfile, contentType: string): SearchContext
    const domainFilters = this.getDomainFiltersForExpertise(expertProfile.primaryExper
    const recencyFilter = this.getRecencyForContentType(contentType);

    return {
      domainFilters,
      recencyFilter,
      expertiseArea: expertProfile.primaryExpertise,
      targetAudience: expertProfile.targetAudience
    };
  }

  private getDomainFiltersForExpertise(expertise: string): string[] {
    const domainMap: Record<string, string[]> = {
      'technology': ['stackoverflow.com', 'github.com', 'techcrunch.com', '-reddit.com
      'healthcare': ['pubmed.ncbi.nlm.nih.gov', 'who.int', 'cdc.gov', '-pinterest.com'
      'finance': ['sec.gov', 'reuters.com', 'bloomberg.com', '-reddit.com'],
      'science': ['nature.com', 'science.org', 'arxiv.org', 'researchgate.net']
    };

    return domainMap[expertise.toLowerCase()] || [];
  }
}
```

3. Sistema de Cache Mais Inteligente

typescript

```typescript
// Baseado nas práticas do ItzCrazyKns/Perplexica
class IntelligentCache {
  private redis: RedisClient;
  private cacheStrategies: Map<string, CacheStrategy>;

  constructor() {
    this.cacheStrategies.set('topic_generation', {
      ttl: 3600, // 1 hora para tópicos
      invalidateOn: ['trending_change', 'expert_profile_update']
    });

    this.cacheStrategies.set('source_validation', {
      ttl: 86400, // 24 horas para validação de fontes
      invalidateOn: ['domain_blacklist_update']
    });

    this.cacheStrategies.set('content_idea', {
      ttl: 1800, // 30 minutos para ideias de conteúdo
      invalidateOn: ['news_update', 'trending_change']
    });
  }

  async get<T>(key: string, type: string): Promise<T | null> {
    const strategy = this.cacheStrategies.get(type);
    if (!strategy) return null;

    const cached = await this.redis.get(key);
    if (!cached) return null;

    const data = JSON.parse(cached);
    const isValid = await this.validateCacheEntry(data, strategy);

    if (!isValid) {
      await this.redis.del(key);
      return null;
    }

    return data.value;
  }
}
```

4. Melhorar Prompt Engineering

typescript

```typescript
// Baseado em estratégias do promptfoo/perplexity
class AdvancedPromptBuilder {
  buildTopicGenerationPrompt(expertProfile: ExpertProfile): string {
    return `
ROLE: Expert Content Strategist specializing in ${expertProfile.primaryExpertise}

CONTEXT:
- Primary Expertise: ${expertProfile.primaryExpertise}
- Target Audience: ${expertProfile.targetAudience}
- Voice Tone: ${expertProfile.voiceTone.join(', ')}
- Content Platforms: ${expertProfile.platforms.join(', ')}

SEARCH INSTRUCTIONS:
1. Focus on information published within the last 7 days
2. Prioritize authoritative sources in ${expertProfile.primaryExpertise}
3. Look for trending discussions, breakthrough announcements, or industry shifts
4. Exclude social media opinions and focus on factual reporting

TASK:
Generate 3 highly relevant content topics that are:
- Currently trending (cite specific recent events/data)
- Aligned with expert's specialization
- Engaging for target audience
- Suitable for ${expertProfile.platforms[0]} format

OUTPUT FORMAT:
For each topic, provide:
1. Compelling title (6-8 words)
2. Brief description with hook angle
3. Recent trend trigger (specific event/statistic with date)
4. 3 authoritative sources with publication dates
5. Engagement potential score (1-10) with justification

Ensure all information is current and cite sources with URLs.`;
  }
}
```

5. Monitoramento e Observabilidade

```typescript
// Inspirado em bastosmichael/perplexity
class PerplexityMonitoring {
  private metrics: MetricsCollector;

  async trackRequest(request: PerplexityRequest, response: PerplexityResponse, duratio
    // Métricas técnicas
    this.metrics.increment('perplexity.requests.total');
    this.metrics.histogram('perplexity.response.duration', duration);
    this.metrics.gauge('perplexity.tokens.used', response.usage.total_tokens);

    // Métricas de qualidade
    if (response.citations) {
      this.metrics.gauge('perplexity.citations.count', response.citations.length);
      await this.validateCitationQuality(response.citations);
    }

    // Métricas de negócio
    const qualityScore = await this.calculateContentQuality(response);
    this.metrics.gauge('content.quality.score', qualityScore);
  }

  private async calculateContentQuality(response: PerplexityResponse): Promise<number>
    const factors = {
      citationCount: response.citations?.length || 0,
      contentLength: response.choices[0].message.content.length,
      sourceReliability: await this.assessSourceReliability(response.citations),
      informationFreshness: await this.assessInformationFreshness(response.citations)
    };

    return this.weightedQualityScore(factors);
  }
}
```

🔧 Problemas de Arquitetura

1. Acoplamento Excessivo

O `PerplexityService` está fazendo muitas responsabilidades. Recomendo separar em:

- `PerplexityClient` (comunicação com API)
- `SearchContextBuilder` (construção de contexto)
- `ResponseProcessor` (processamento de respostas)
- `QualityAssessor` (avaliação de qualidade)

2. Falta de Fallback Strategy

Implementar fallback para Anthropic quando Perplexity falhar:

```typescript
class HybridSearchService {
  async searchAndGenerate(params: SearchParams): Promise<SearchResponse> {
    try {
      return await this.perplexityService.search(params);
    } catch (error) {
      if (this.isRateLimitError(error) || this.isServiceUnavailable(error)) {
        console.warn('Perplexity unavailable, falling back to Anthropic');
        return await this.anthropicService.search(params);
      }
      throw error;
    }
  }
}
```

📊 KPIs Adicionais Recomendados

1. **Citation Accuracy Rate**: % de URLs válidas e acessíveis
2. **Source Authority Score**: Média de confiabilidade das fontes
3. **Information Recency Score**: Quão recentes são as informações
4. **Domain Filter Effectiveness**: Taxa de conteúdo relevante vs irrelevante
5. **Cost per Quality Point**: Custo por token vs qualidade do conteúdo

🎯 Próximos Passos Prioritários

1. **Semana 1**: Implementar `search_domain_filter` e filtros de recência
2. **Semana 2**: Desenvolver sistema de cache inteligente
3. **Semana 3**: Implementar validação robusta de fontes
4. **Semana 4**: Adicionar monitoramento e métricas de qualidade

💡 Considerações Finais

A implementação atual é um bom ponto de partida, mas precisa ser mais robusta para produção. As funcionalidades de filtro de domínios, rate limiting inteligente e validação de fontes são essenciais para entregar valor real aos usuários do ExpertPlanner.

Recomendo fortemente estudar os repositórios `japelsin/pplx` e `RossH121/perplexity-mcp` como referência para implementações mais maduras da API Perplexity.