

SISTEMA DE EXTRAÇÃO DE DADOS FISCAIS

Relatório Técnico - Arquitetura Multiagente com IA

Grupo: Agente Aprende

Integrantes:

- Pedro Markovicz



Sumário Executivo

O Sistema de Extração de Dados Fiscais representa uma solução multiagente inovadora desenvolvida para automatizar integralmente o processamento de Notas Fiscais Eletrônicas (NF-e) brasileiras. A solução implementa uma arquitetura sofisticada baseada em **LangGraph** e **LangChain**, framework especializado em orquestração de agentes de IA, processando documentos fiscais em formatos **XML** e **PDF (DANFE)** com alta precisão e confiabilidade.

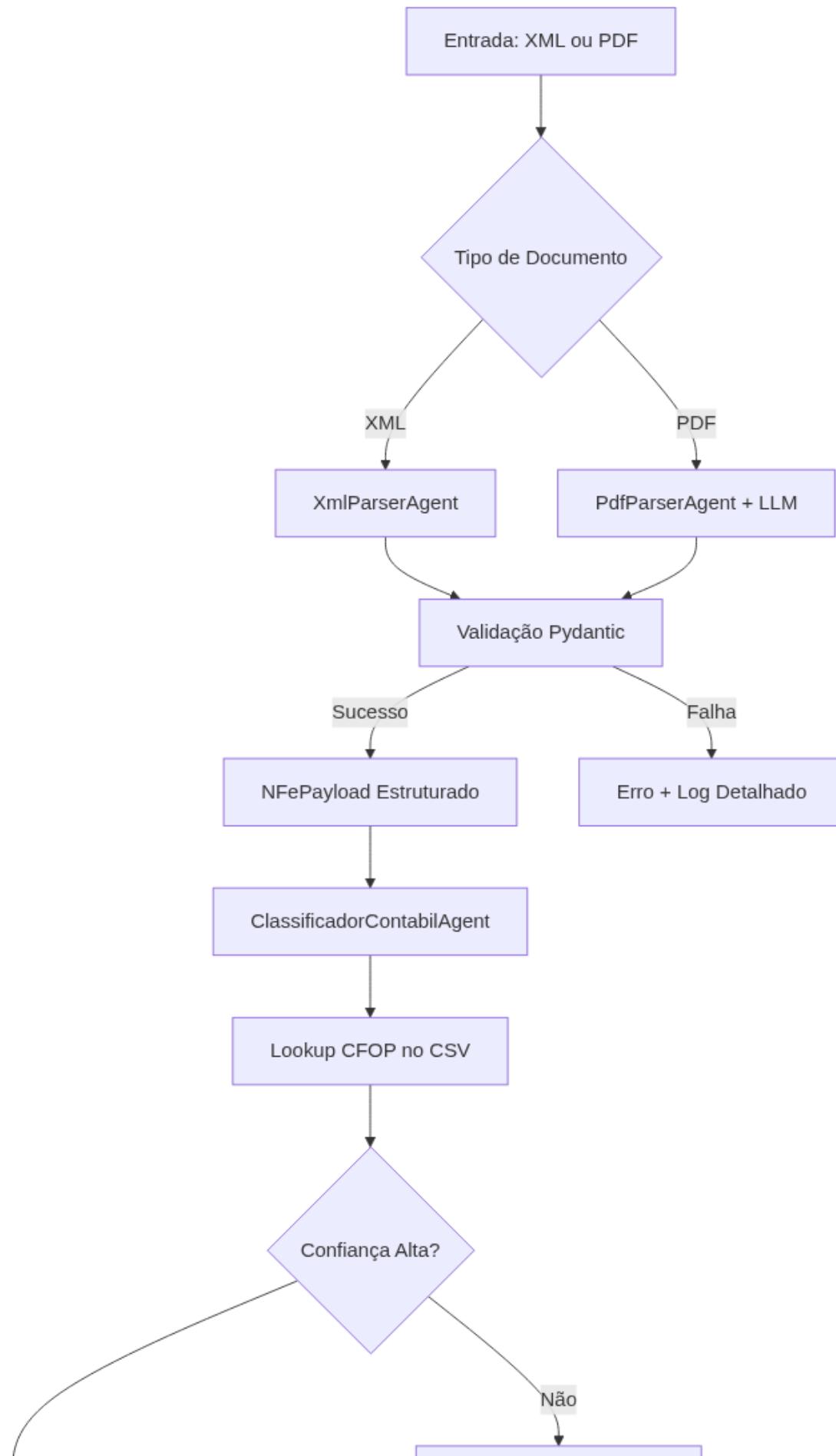
O sistema caracteriza-se por uma arquitetura avançada que combina três agentes especializados: **XmlParserAgent** (parsing direto de XML), **PdfParserAgent** (extração via LLM multi-provedor), e **ClassificadorContabilAgent** (classificação fiscal e contábil), integrados através de um pipeline determinístico com estado compartilhado tipado. A solução processa automaticamente documentos fiscais complexos, aplicando regras contábeis brasileiras e validações fiscais rigorosas (CFOP, NCM, CST/CSOSN, impostos ICMS/IPI/PIS/COFINS).

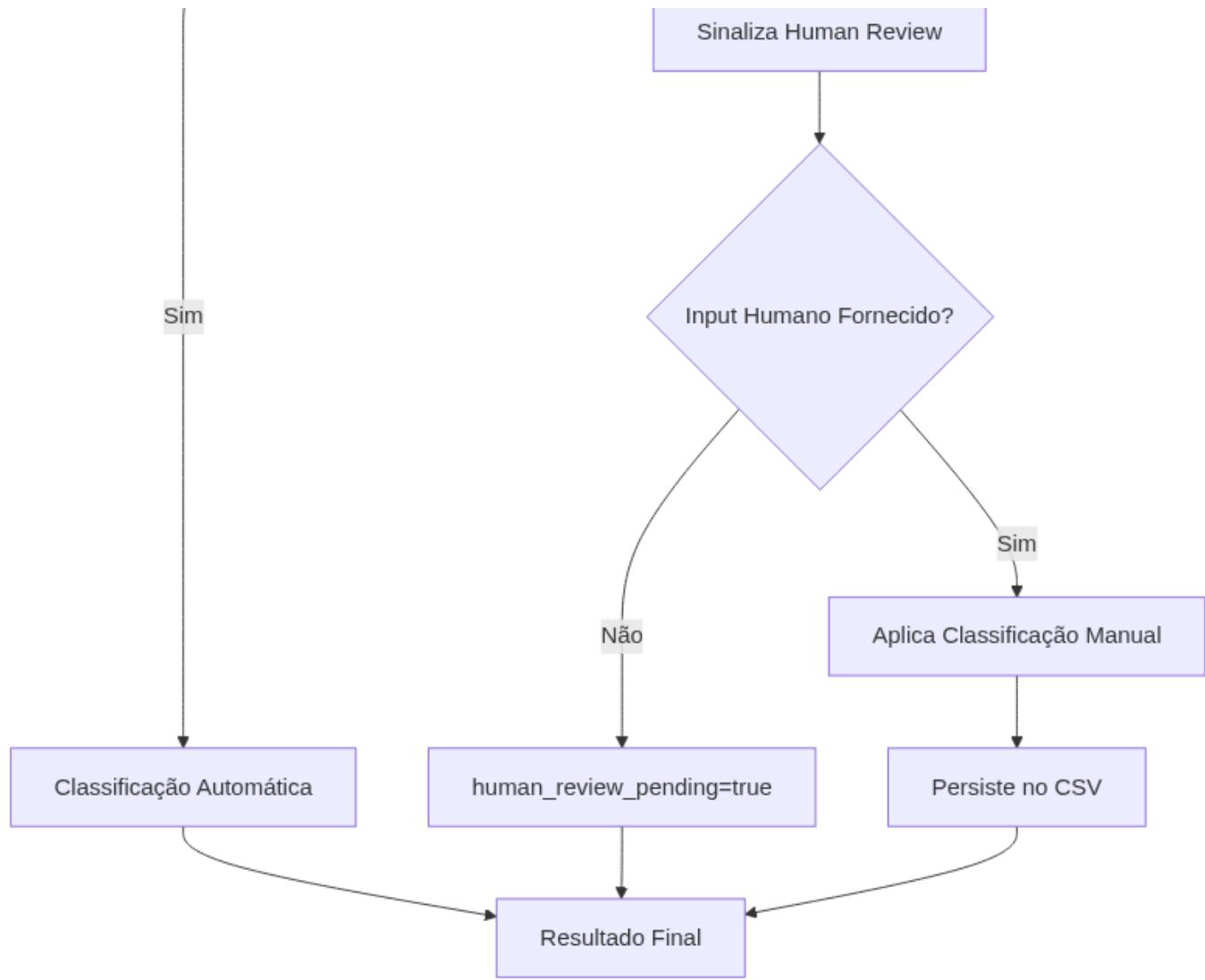
Os principais benefícios identificados incluem **automação completa da extração de dados**, eliminação de erros humanos em interpretações fiscais complexas, suporte a múltiplos provedores de IA (OpenAI, Google Gemini, Groq), e garantia de conformidade com regras fiscais específicas. O sistema utiliza tecnologias modernas como **Python 3.10+**, **FastAPI**, **Streamlit**, **UV** para gerenciamento de dependências, e implementa validação completa com **Pydantic**, observabilidade com logs estruturados e rastreabilidade de estado.

A arquitetura contempla **human-in-the-loop** para casos de baixa confiança, persistência de aprendizado em CSV, interface web moderna e API REST completa, tornando-se uma solução enterprise-ready para automação contábil.

1. Arquitetura e Visão Geral da Solução

1.1 Workflow Sequencial LangGraph





Características do Workflow

- **SEQUENCIAL E DETERMINÍSTICO:** Cada agente depende do anterior, mesma entrada produz sempre mesma saída
- **ESTADO TIPADO E IMUTÁVEL:** TypedDict com Pydantic garantindo integridade de tipos em cada transição
- **MULTI-FORMATO:** Suporte nativo a XML (parsing direto) e PDF (extração via LLM)
- **RASTREÁVEL:** Histórico completo de transformações com logs estruturados
- **OBSERVÁVEL:** Métricas e estado compartilhado em tempo real via LangGraph
- **RESILIENTE:** Falha rápida com diagnóstico detalhado e tratamento de erros por camada
- **HUMAN-IN-THE-LOOP:** Revisão humana integrada para casos de baixa confiança

1.2 Fluxo de Dados Detalhado

O sistema implementa um **fluxo sequencial robusto** onde cada agente possui dependência completa do anterior, garantindo integridade e rastreabilidade dos dados:

Etapa 1 - Ingestão e Parsing:

- **Entrada XML:** Leitura direta via `xmldict` com tratamento de namespaces, sanitização de campos (CFOP, NCM, valores monetários), extração de estruturas aninhadas (emitente, destinatário, itens, impostos)

- **Entrada PDF:** Extração de texto via PyMuPDF com fallback OCR (Tesseract), processamento via LLM multi-provedor (OpenAI GPT-4o-mini, Google Gemini 1.5 Pro, Groq Llama 3.1), sanitização e normalização de output JSON

Etapa 2 - Validação Pydantic:

- Modelos imutáveis (frozen=True) com tipagem forte e field validators
- Normalização automática (CNPJ/CPF, CEP, telefone, valores monetários brasileiros)
- Validações fiscais específicas (CFOP 4 dígitos, NCM 8 dígitos, CEST 7 dígitos, UF enum com 27 estados)
- Validação de impostos (CST/CSOSN, origem, alíquotas, bases de cálculo)

Etapa 3 - Classificação Contábil:

- Lookup em CSV mestre ([contas_por_cfop.csv](#)) por (CFOP, regime tributário)
- Determinação automática de natureza de operação (interna vs. interestadual)
- Scoring de confiança baseado em regras e histórico
- Fallback inteligente por prefixo CFOP (1/2=entrada, 5/6=saída)

Etapa 4 - Human Review (Condisional):

- Ativação automática quando confiança < 0.75 ou CFOP não mapeado
- Interface Streamlit integrada para revisão manual
- Persistência de aprendizado no CSV mestre
- Reaplicação do workflow com classificação humana

Etapa 5 - Resultado Final:

- JSON estruturado com payload completo da NF-e
- Classificação contábil (débito/crédito, justificativa, confiança)
- Metadados de processamento (timestamps, versões, needs_review)
- Exportação para integração com ERPs

1.3 Características Técnicas Fundamentais

Framework Base: A solução utiliza **LangGraph 0.2+** combinado com **LangChain** para orquestração robusta de agentes especializados. A escolha do LangGraph foi motivada pela necessidade de:

- Execução determinística com grafo de estados
- Estado compartilhado tipado via TypedDict
- Roteamento condicional baseado em dados de estado
- Observabilidade nativa e rastreamento de execução
- Integração nativa com provedores de LLM

Estado Compartilhado: O sistema implementa a classe **WorkflowState** baseada em **TypedDict**, garantindo:

- **Imutabilidade lógica** (cada nó retorna novo estado atualizado)
- **Tipagem forte** (validação em tempo de desenvolvimento e execução)
- **Rastreabilidade** (histórico completo disponível no estado final)
- **Observabilidade** (inspeção de estado em qualquer ponto do workflow)

Validação com Pydantic: Todos os modelos de domínio usam **Pydantic 2.x** com:

- Modelos frozen (imutáveis pós-criação)
 - Field validators para normalização automática
 - Model validators para validações cruzadas
 - Aliases para mapeamento XML/JSON → Python
 - Enums para valores controlados (UF, regimes)
-

2. Especificação dos Agentes Inteligentes

2.1 XmlParserAgent

| Atributo | Detalhes |
|---------------------|--|
| Localização | <code>src/agents/xml_parser_agent.py</code> |
| Posição no Workflow | Primeiro agente executado (alternativa ao PDF) |
| Entrada | Arquivo XML da NF-e (formato SEFAZ) |
| Saída | <code>NFePayload</code> validado via Pydantic |

Responsabilidades Específicas:

- Lê bytes do arquivo XML e realiza parsing com `xmldict`
- Tenta localizar nó `infNFe` em caminhos comuns (`nfeProc.NFe.infNFe`, `NFe.infNFe`)
- Implementa **dupla tentativa de parsing**: com namespaces e removendo namespaces comuns
- Extrai **dados completos do emitente**: razão social, CNPJ, IE, endereço completo (logradouro, número, bairro, município, UF, CEP, telefone)
- Extrai **dados completos do destinatário**: razão social, CNPJ/CPF, IE, indIEDest, endereço completo
- Processa **itens da nota** com campos: descrição, NCM, CEST, valor, quantidade, valor unitário, unidade comercial, código do produto
- Extrai **impostos por item**: ICMS (CST/CSOSN, origem, base de cálculo, alíquota, valor), IPI, PIS, COFINS
- Extrai **totais de impostos**: vBC, vICMS, vIPI, vPIS, vCOFINS do bloco `total.ICMSTot`

Validações Implementadas:

- Remoção de NCM inválido (não 8 dígitos) antes da validação Pydantic
- Remoção de CEST inválido (não 7 dígitos)
- Normalização de CNPJ/CPF para apenas dígitos
- Tratamento de IE especial (ISENTO)
- Garantia de xProd mínimo ("Item" como fallback)
- Conversão de valores monetários brasileiros (vírgula → ponto)

Estratégia de Parsing:

1. Primeira tentativa: parsing direto preservando namespaces
2. Segunda tentativa: remoção de namespaces comuns via regex
3. Localização de `infNFe` em múltiplos caminhos possíveis
4. Tratamento de nós que podem ser dict ou list (`_as_list` helper)
5. Acesso seguro a chaves aninhadas (`safe_get` helper)

Exemplo de Log de Sucesso:

```
[INFO] NFe parse OK | cfop=5102 emitente=EMPRESA XYZ LTDA (CNPJ: 12345678000195,
UF: SP)
                           destinatario=CLIENTE ABC (CNPJ: 98765432000156, UF: RJ) itens=3
                           vtotal=1500.00
```

2.2 PdfParserAgent

| Atributo | Detalhes |
|----------------------------|---|
| Localização | <code>src/agents/pdf_parser_agent.py</code> |
| Posição no Workflow | Primeiro agente executado (alternativa ao XML) |
| Entrada | Arquivo PDF do DANFE (Documento Auxiliar da NF-e) |
| Saída | <code>NFePayload</code> validado via Pydantic |

Responsabilidades Específicas:

- Extrai texto de PDF via **PyMuPDF (fitz)** com análise de camada de texto
- Implementa **OCR automático** via Tesseract quando PDF não tem texto (imagens escaneadas)
- Utiliza **LLM multi-provedor** para extração estruturada de dados fiscais
- Suporta **OpenAI** (gpt-4o-mini), **Google Gemini** (gemini-1.5-pro), **Groq** (llama-3.1-70b-versatile)
- Aplica **sanitização rigorosa** do output LLM antes da validação Pydantic
- Normaliza números brasileiros (1.234,56 → 1234.56)
- Extrai impostos quando disponíveis no layout do PDF

Configuração via Variáveis de Ambiente (.env):

```
# Provedor de LLM
PDF_LLM_PROVIDER=openai # openai | gemini | groq

# Chaves de API (conforme provedor escolhido)
OPENAI_API_KEY=sk-...
GOOGLE_API_KEY=AIza...
GROQ_API_KEY=gsk...

# Modelo específico (opcional, tem defaults)
PDF_LLM_MODEL=gpt-4o-mini # ou gemini-1.5-pro, llama-3.1-70b-versatile

# Temperatura (0.0 = determinístico)
PDF_LLM_TEMPERATURE=0.0
```

Prompts Especializados:

- Schema JSON detalhado com tipos, padrões regex e descrições
- Instruções explícitas sobre campos obrigatórios vs. opcionais
- Orientação sobre CNPJ vs. CPF (mutuamente exclusivo)
- Atenção especial para IE do destinatário (não confundir com IE do emitente)
- Instruções sobre impostos (incluir apenas se visíveis no PDF)
- Formato de saída: JSON puro sem markdown

Fluxo de Processamento PDF:

1. **Extração de Texto:** PyMuPDF tenta ler camada de texto
2. **OCR Fallback:** Se texto < 20 chars, ativa OCR via Tesseract
3. **Análise Heurística:** Detecta chave de acesso, valor total, CFOPs, UFs via regex e posicionamento
4. **Extração via LLM:** Envia texto + schema JSON para o provedor configurado
5. **Sanitização:** Normaliza CFOP (4 dígitos), UFs (uppercase), NCM/CEST (validação), números (vírgula→ponto)
6. **Validação Pydantic:** Garante conformidade total com modelos de domínio

Tratamento de Campos Complexos:

- **Emitente:** Extração completa (razão social, CNPJ 14 dígitos, IE, UF, município, endereço, CEP, telefone)
- **Destinatário:** Lógica CNPJ vs. CPF (11 ou 14 dígitos), tratamento de IE do destinatário separado do emitente
- **Itens:** Descrição, NCM, CEST, valor, quantidade, valor unitário, unidade, código
- **Impostos:** ICMS (CST/CSOSN), IPI (opcional), PIS, COFINS (quando disponíveis no PDF)

Exemplo de Output Saneado:

```
{
  "cfop": "5102",
  "emitente": {
    "xNome": "EMPRESA EMITENTE LTDA",
    "CNPJ": "12345678000195",
    "uf": "SP"
    # ... demais campos
  },
  "destinatario": {
    "xNome": "CLIENTE DESTINATARIO",
    "CPF": "12345678901", # ou CNPJ
    "uf": "RJ"
    # ... demais campos
  },
  "valor_total": 1500.00,
  "itens": [...]
}
```

2.3 ClassificadorContabilAgent

| Atributo | Detalhes |
|---------------------|--|
| Localização | <code>src/agents/classificador_contabil_agent.py</code> |
| Posição no Workflow | Segundo agente executado (após parser) |
| Entrada | <code>NFePayload</code> validado + regime tributário (opcional) |
| Saída | <code>ClassificacaoContabil</code> (contas débito/crédito, justificativa, confiança) |

Responsabilidades Específicas:

- Carrega mapa de classificação contábil de `data_sources/contas_por_cfop.csv` (cacheable com `@lru_cache`)
- Executa **lookup por (CFOP, regime tributário)** com fallback para wildcard (*)
- Determina **natureza de operação**: interna (mesma UF) vs. interestadual (UFs diferentes)
- Calcula **score de confiança** baseado na fonte do mapeamento (CSV exato, CSV wildcard, fallback)
- Implementa **fallback inteligente** por prefixo CFOP quando não encontrado no CSV
- Sinaliza **necessidade de revisão humana** quando confiança < 0.75 ou CFOP não mapeado

Estrutura do CSV Mestre:

```
cfop,regime,conta_debito,conta_credito,justificativa_base,confianca
5102,*,Clientes,Receita de Vendas,Venda de mercadoria adquirida ou recebida de terceiros,0.90
1102,*,Estoques de Mercadorias,Fornecedores,Compra para comercialização,0.90
5101,simples,Clientes,Receita de Vendas - Simples,Venda de produção do estabelecimento,0.95
```

Lógica de Matching:

1. **Tentativa 1:** Match exato por (CFOP, regime específico) → confiança alta (0.90-0.95)
2. **Tentativa 2:** Match por (CFOP, regime=*) → confiança média (0.70-0.80)
3. **Fallback:** Classificação por prefixo CFOP → confiança baixa (0.50-0.65)
 - `1xxx/2xxx`: Entrada (compra) → Débito: Estoques / Crédito: Fornecedores
 - `5xxx/6xxx`: Saída (venda) → Débito: Clientes / Crédito: Receita

Human-in-the-Loop:

- **Trigger:** `confianca < 0.75` ou CFOP não encontrado
- **Ação:** Seta `needs_human_review=True` e `review_reason` com justificativa
- **Workflow:** Grafo roteia para `human_review_pending=True` se não houver input humano no estado
- **Aprendizado:** Input humano é persistido no CSV via `upsert_cfop_mapping()`
- **Reaplicação:** Workflow reprocessa com nova classificação manual aplicada

Exemplo de Classificação Automática:

```
ClassificacaoContabil(
    cfop="5102",
    natureza_operacao="interna", # SP → SP
    conta_debito="Clientes",
    conta_credito="Receita de Vendas",
    justificativa="Venda de mercadoria adquirida ou recebida de terceiros. "
                    "Natureza: interna. Valor total da NF-e considerado para contexto: 1500.00.",
    ncm_itens=[ "12345678", None, "87654321"],
    confianca=0.90,
    needs_human_review=False,
    review_reason=None,
    rule_version="v0.4"
)
```

Exemplo de Sinalização de Revisão:

```
ClassificacaoContabil(  
    cfop="6949",  
    natureza_operacao="interestadual", # SP → MG  
    conta_debito="Conta a Classificar (Débito)",  
    conta_credito="Conta a Classificar (Crédito)",  
    justificativa="CFOP fora dos intervalos mínimos do MVP; aplicar regras  
detalhadas."  
        "Natureza: interestadual. Valor total da NF-e considerado para  
contexto: 2500.00.",  
    ncm_itens=[...],  
    confianca=0.50,  
    needs_human_review=True,  
    review_reason="Mapeamento não encontrado no CSV para CFOP 6949  
(regime=simples)."  
        "Aplicado fallback por prefixo. Revisão humana obrigatória.",  
    rule_version="v0.4"  
)
```

3. Tecnologias e Implementação

3.1 Stack Tecnológico

| Categoria | Tecnologia | Versão | Justificativa |
|-------------------------------|-------------------------|--------|--|
| Linguagem Principal | Python | 3.10+ | Tipagem forte moderna, performance otimizada, ecossistema IA/ML maduro |
| Gerenciamento de Dependências | UV | Latest | Substituto moderno ao pip/poetry, instalação 10-100x mais rápida, resolução determinística de dependências |
| Validação de Dados | Pydantic | 2.x | Validação automática, modelos imutáveis, serialização JSON, field validators customizados |
| Orquestração de Agentes | LangGraph | 0.2+ | Grafo de estados tipado, execução determinística, roteamento condicional, observabilidade nativa |
| Integração LLM | LangChain | 0.3+ | Multi-provedor (OpenAI, Gemini, Groq), prompts estruturados, output parsers, retry logic |
| Processamento de Dados | Pandas | 2.x | Manipulação de DataFrames, análise exploratória, exportação CSV/Excel |
| Parsing XML | xmldict | 0.13+ | Conversão XML→dict pythônico, namespace handling, leitura simplificada |
| Processamento PDF | PyMuPDF (fitz) | 1.24+ | Extração de texto, renderização de páginas, análise de layout, performance superior |
| OCR | Tesseract + pytesseract | 5.x | OCR gratuito e robusto, suporte a português, fallback para PDFs escaneados |
| API REST | FastAPI | 0.115+ | Performance assíncrona, validação automática via Pydantic, docs OpenAPI/Swagger integradas |
| Interface Web | Streamlit | 1.38+ | Desenvolvimento rápido de UIs, reatividade nativa, componentes rich (tabs, expanders, metrics) |
| Testes | pytest | 8.x | Framework moderno, fixtures poderosas, parametrização, cobertura de código |

3.2 Arquitetura LangGraph

Orquestração: LangGraph implementa padrão **DAG (Directed Acyclic Graph)** com roteamento condicional garantindo:

- Execução determinística (mesma entrada → mesma saída)
- Rastreabilidade completa (estado acessível em qualquer ponto)
- Estado compartilhado tipado via `TypedDict` (`WorkflowState`)
- Transições baseadas em dados (roteamento condicional após classificador)
- Observabilidade nativa (logs, traces, estado intermediário)

Configuração do Workflow (`src/workflow/graph.py`):

```

from langgraph.graph import StateGraph, END
from src.workflow.state import WorkflowState
from src.workflow.nodes import xml_parser_node, classificador_contabil_node,
human_review_node

def _route_after_classificador(state: WorkflowState) -> str:
    """Roteamento condicional baseado em necessidade de revisão humana."""
    needs = bool(state.get("classificacao_needs_review"))
    has_input = bool(state.get("human_review_input"))

    if not needs:
        return "done" # Classificação OK, finaliza
    if needs and not has_input:
        state["human_review_pending"] = True
        return "need_input" # Aguarda input humano, finaliza temporariamente
    return "human_review" # Aplica revisão humana e finaliza

def build_graph():
    graph = StateGraph(WorkflowState)

    # Definição de nós
    graph.add_node("xml_parser", xml_parser_node)
    graph.add_node("classificador_contabil", classificador_contabil_node)
    graph.add_node("human_review", human_review_node)

    # Fluxo principal
    graph.set_entry_point("xml_parser")
    graph.add_edge("xml_parser", "classificador_contabil")

    # Roteamento condicional
    graph.add_conditional_edges(
        "classificador_contabil",
        _route_after_classificador,
        {
            "done": END,
            "need_input": END,
            "human_review": "human_review",
        },
    )

    graph.add_edge("human_review", END)

    return graph.compile()

```

Estado Compartilhado ([src/workflow/state.py](#)):

```

from typing import TypedDict, Optional, Dict, Any

class WorkflowState(TypedDict, total=False):
    # Inputs

```

```

xml_path: str                      # Caminho do XML
pdf_path: str                        # Caminho do PDF
regime_tributario: str              # "simples" | "presumido" | "real"

# Outputs do parser
ok: bool                            # Sucesso/falha do parsing
payload: Dict[str, Any]            # NFePayload serializado
error: Optional[str]               # Mensagem de erro se falhar

# Outputs do classificador
classificacao_ok: bool            # Sucesso da classificação
classificacao: Dict[str, Any]      # ClassificacaoContabil serializada
classificacao_needs_review: bool   # Requer revisão humana?
classificacao_review_reason: Optional[str] # Motivo da revisão

# Human review
human_review_input: Dict[str, Any] # Input manual (cfop, contas, etc.)
human_review_pending: bool        # Aguardando input?
human_review_applied: bool       # Input aplicado?

```

3.3 Padrões de Implementação

Estado Imutável e Funcional

Cada nó do grafo implementa **padrão funcional puro**:

- Recebe estado atual como input (somente leitura)
- Retorna novo estado atualizado (imutável)
- Garante rastreabilidade completa (estado anterior preservado)
- Permite rollback e replay de transições
- Persiste metadados de processamento (timestamps, versões)

Exemplo de Nó:

```

def xml_parser_node(state: WorkflowState) -> WorkflowState:
    """Parser de XML: recebe estado com xml_path, retorna estado com payload."""
    xml_path = state.get("xml_path")
    if not xml_path:
        # Tenta PDF se XML não fornecido
        return pdf_parser_node(state)

    try:
        payload = parse_xml(xml_path)
        return {
            **state,
            "ok": True,
            "payload": payload.model_dump(),
            "error": None
        }
    except XmlParseError as e:
        return {

```

```
    **state,  
    "ok": False,  
    "payload": None,  
    "error": str(e)  
}
```

Tratamento de Erros

Sistema implementa **defesa em profundidade**:

- **Captura granular** por agente com contexto específico
- **Logging estruturado** com níveis (DEBUG, INFO, WARNING, ERROR)
- **Estratégia de falha rápida** com diagnóstico detalhado
- **Validação em múltiplas camadas**: sanitização → parsing → validação Pydantic
- **Retry logic** para chamadas LLM (via LangChain)

Exemplo de Erro Tratado:

```
# Em pdf_parser_agent.py  
try:  
    llm = _get_llm()  
    template, schema_hint = _build_prompt()  
    parser = JsonOutputParser()  
    chain = template | llm | parser  
    result = chain.invoke({'document': text[:150000], 'schema':  
json.dumps(schema_hint)})  
    sanitized = _sanitize_llm_payload(result)  
    payload = NFePayload.model_validate(sanitized)  
    return payload  
except Exception as e:  
    logger.exception('Falha ao extrair payload com LLM')  
    raise XmlParseError(f'Falha na extração via LLM: {e}') from e
```

4. Modelos de Domínio e Validação

4.1 Modelos Pydantic Principais

NFePayload (Estrutura Raiz)

```
class NFePayload(BaseModel):
    model_config = ConfigDict(frozen=True, populate_by_name=True)

    cfop: str = Field(pattern=r"^\d{4}$")                                # CFOP obrigatório (4 dígitos)
    emitente: Emitente                                                    # Dados completos do emitente
    destinatario: Destinatario                                              # Dados completos do destinatário
    valor_total: float = Field(ge=0)                                         # Valor total ≥ 0
    itens: List[NFeItem] = Field(min_length=1)                               # Ao menos 1 item
    totais_impostos: Optional[TotaisImpostos] = None                      # Totais consolidados (opcional)

    @property
    def emitente_uf(self) -> UfEnum:
        """Property para compatibilidade retroativa."""
        return self.emitente.uf

    @property
    def destinatario_uf(self) -> UfEnum:
        """Property para compatibilidade retroativa."""
        return self.destinatario.uf
```

Emitente (Pessoa Jurídica Emissora)

```
class Emitente(BaseModel):
    model_config = ConfigDict(frozen=True, populate_by_name=True)

    # Campos ALTA prioridade
    razao_social: str = Field(alias="xNome", min_length=1)
    cnpj: str = Field(alias="CNPJ", pattern=r"^\d{14}$")

    # Campos MEDIA prioridade
    inscricao_estadual: Optional[str] = Field(alias="IE", default=None)
    uf: UfEnum
    municipio: Optional[str] = Field(alias="xMun", default=None)
    bairro: Optional[str] = Field(alias="xBairro", default=None)
    logradouro: Optional[str] = Field(alias="xLgr", default=None)
    numero: Optional[str] = Field(alias="nro", default=None)

    # Campos BAIXA prioridade
```

```
cep: Optional[str] = Field(alias="CEP", default=None, pattern=r"^\d{8}$")
telefone: Optional[str] = Field(alias="fone", default=None)
```

Destinatario (Pessoa Física ou Jurídica Receptora)

```
class Destinatario(BaseModel):
    model_config = ConfigDict(frozen=True, populate_by_name=True)

    razao_social: str = Field(alias="xNome", min_length=1)
    cnpj: Optional[str] = Field(alias="CNPJ", default=None, pattern=r"^\d{14}$")
    cpf: Optional[str] = Field(alias="CPF", default=None, pattern=r"^\d{11}$")
    inscricao_estadual: Optional[str] = Field(alias="IE", default=None)
    indicador_ie: Optional[str] = Field(alias="indIEDest", default=None)
    uf: UfEnum
    # ... demais campos de endereço ...

    @model_validator(mode="after")
    def validate_cpf_or_cnpj(self):
        """Garante mutuamente exclusivo: CPF XOR CNPJ."""
        if not self.cpf and not self.cnpj:
            raise ValueError("Destinatario deve ter CPF ou CNPJ")
        if self.cpf and self.cnpj:
            raise ValueError("Destinatario nao pode ter CPF e CNPJ
simultaneamente")
        return self
```

NFeItem (Produto/Serviço da Nota)

```
class NFeItem(BaseModel):
    model_config = ConfigDict(frozen=True, populate_by_name=True)

    descricao: str = Field(alias="xProd", min_length=1)
    ncm: Optional[str] = Field(alias="NCM", default=None, pattern=r"^\d{8}$")
    cest: Optional[str] = Field(alias="CEST", default=None, pattern=r"^\d{7}$")
    valor: float = Field(alias="vProd", ge=0)

    # Campos comerciais
    quantidade: Optional[float] = Field(alias="qCom", default=None, gt=0)
    valor_unitario: Optional[float] = Field(alias="vUnCom", default=None, gt=0)
    unidade_comercial: Optional[str] = Field(alias="uCom", default=None)
    codigo_produto: Optional[str] = Field(alias="cProd", default=None)

    # Impostos do item
    impostos: Optional[ImpostosItem] = None

    @model_validator(mode="after")
    def validate_calculation(self):
        """Valida quantidade * valor_unitario ≈ valor (tolerância 2 centavos)."""
        pass
```

```

    if self.quantidade is not None and self.valor_unitario is not None:
        calculated = self.quantidade * self.valor_unitario
        difference = abs(calculated - self.valor)
        if difference > 0.02:
            logger.warning("Divergência de cálculo: %.4f * %.4f = %.2f != %.2f",
                           self.quantidade, self.valor_unitario, calculated,
                           self.valor)
    return self

```

4.2 Modelos de Impostos

ICMS (Imposto sobre Circulação de Mercadorias e Serviços)

```

class ICMS(BaseModel):
    model_config = ConfigDict(frozen=True, populate_by_name=True)

    # CST (Regime Normal) OU CSOSN (Simples Nacional)
    cst: Optional[str] = Field(alias="CST", default=None, pattern=r"^\d{2}$")
    csosn: Optional[str] = Field(alias="CSOSN", default=None, pattern=r"^\d{3}$")

    orig: Optional[str] = Field(alias="orig", default=None, pattern=r"^[0-8]$")
    v_bc: Optional[float] = Field(alias="vBC", default=None, ge=0)
    p_icms: Optional[float] = Field(alias="pICMS", default=None, ge=0)
    v_icms: Optional[float] = Field(alias="vICMS", default=None, ge=0)
    mod_bc: Optional[str] = Field(alias="modBC", default=None)

    @model_validator(mode="after")
    def validate_cst_or_csosn(self):
        """Exatamente um deve estar presente: CST XOR CSOSN."""
        if not self.cst and not self.csosn:
            raise ValueError("ICMS deve ter CST ou CSOSN")
        if self.cst and self.csosn:
            raise ValueError("ICMS não pode ter CST e CSOSN simultaneamente")
        return self

```

ImpostosItem (Consolidação de Impostos por Item)

```

class ImpostosItem(BaseModel):
    model_config = ConfigDict(frozen=True, populate_by_name=True)

    icms: ICMS                      # ICMS obrigatório
    ipi: Optional[IPI] = None          # IPI opcional (nem todos produtos têm)
    pis: Optional[PIS] = None          # PIS opcional (PDFs podem não mostrar)
    cofins: Optional[COFINS] = None    # COFINS opcional (PDFs podem não mostrar)

```

TotaisImpostos (Consolidação de Totais da NF-e)

```
class TotaisImpostos(BaseModel):
    model_config = ConfigDict(frozen=True, populate_by_name=True)

    v_bc_icms: Optional[float] = Field(alias="vBC", default=None, ge=0)
    v_icms: Optional[float] = Field(alias="vICMS", default=None, ge=0)
    v_ipi: Optional[float] = Field(alias="vIPI", default=None, ge=0)
    v_pis: Optional[float] = Field(alias="vPIS", default=None, ge=0)
    v_cofins: Optional[float] = Field(alias="vCOFINS", default=None, ge=0)
```

4.3 Enum UfEnum (Estados Brasileiros)

```
class UfEnum(str, Enum):
    AC = "AC"; AL = "AL"; AP = "AP"; AM = "AM"; BA = "BA"; CE = "CE"; DF = "DF"
    ES = "ES"; GO = "GO"; MA = "MA"; MT = "MT"; MS = "MS"; MG = "MG"; PA = "PA"
    PB = "PB"; PR = "PR"; PE = "PE"; PI = "PI"; RJ = "RJ"; RN = "RN"; RS = "RS"
    RO = "RO"; RR = "RR"; SC = "SC"; SP = "SP"; SE = "SE"; TO = "TO"
```

4.4 Field Validators e Normalizações

CNPJ/CPF: Remoção de caracteres não numéricos

```
@field_validator("cnpj", mode="before")
@classmethod
def _normalize_cnpj(cls, v: Any) -> str:
    if v is None: return ""
    if isinstance(v, str): return "".join(filter(str.isdigit, v))
    return str(v)
```

Inscrição Estadual: Tratamento de "ISENTO"

```
@field_validator("inscricao_estadual", mode="before")
@classmethod
def _normalize_ie(cls, v: Any) -> Optional[str]:
    if v is None or v == "": return None
    v_upper = str(v).strip().upper()
    if "ISENT" in v_upper: return "ISENTO"
    return v_upper
```

Valores Monetários: Conversão de vírgula para ponto

```
@field_validator("valor_total", mode="before")
@classmethod
def _normalize_valor_total(cls, v: Any) -> Any:
```

```
if isinstance(v, str): return v.replace(",", ".")  
return v
```

5. API REST e Interface Web

5.1 Endpoints FastAPI (`src/api/main.py`)

Healthcheck:

- `GET /health` → Retorna `{"status": "ok"}`

Classificação (sem revisão humana):

- `POST /classificar/path` → JSON: `{"xml_path": "caminho/arquivo.xml"}`
- `POST /classificar/pdf_path` → JSON: `{"pdf_path": "caminho/arquivo.pdf"}`
- `POST /classificar/xml` → Upload de arquivo XML
- `POST /classificar/pdf` → Upload de arquivo PDF

Revisão Humana:

- `POST /classificar/review/path` → XML via caminho + JSON de revisão
- `POST /classificar/review/pdf_path` → PDF via caminho + JSON de revisão
- `POST /classificar/review/xml` → Upload XML + JSON de revisão
- `POST /classificar/review/pdf` → Upload PDF + JSON de revisão

Gestão de Mapeamentos:

- `PUT /classificar/cfop-mapping` → Upsert no CSV (cfop, regime, contas, justificativa, confiança)

Exemplo de Requisição de Classificação:

```
curl -X POST "http://localhost:8000/classificar/xml" \
-F "xml_file=@data/exemplos/xml/nfe_exemplo_1.xml"
```

Exemplo de Response de Sucesso:

```
{
  "ok": true,
  "payload": {
    "cfop": "5102",
    "emitente": {
      "razao_social": "EMPRESA XYZ LTDA",
      "cnpj": "12345678000195",
      "uf": "SP"
    },
    "destinatario": {
      "razao_social": "CLIENTE ABC",
      "cnpj": "98765432000156",
      "uf": "RJ"
    },
    "valor_total": 1500.00,
    "itens": [...]
  },
}
```

```

"classificacao_ok": true,
"classificacao": {
    "cfop": "5102",
    "natureza_operacao": "interestadual",
    "conta_debito": "Clientes",
    "conta_credito": "Receita de Vendas",
    "justificativa": "Venda de mercadoria adquirida ou recebida de terceiros.
Natureza: interestadual. Valor total da NF-e considerado para contexto: 1500.00.",
    "confianca": 0.90,
    "needs_human_review": false,
    "review_reason": null
},
"human_review_pending": false
}

```

Exemplo de Response com Revisão Necessária:

```

{
    "ok": true,
    "payload": {...},
    "classificacao_ok": true,
    "classificacao": {
        "cfop": "6949",
        "natureza_operacao": "interestadual",
        "conta_debito": "Conta a Classificar (Débito)",
        "conta_credito": "Conta a Classificar (Crédito)",
        "confianca": 0.50,
        "needs_human_review": true,
        "review_reason": "Mapeamento não encontrado no CSV para CFOP 6949
(regime=simples). Aplicado fallback por prefixo. Revisão humana obrigatória."
    },
    "human_review_pending": true
}

```

5.2 Interface Streamlit ([src/app/streamlit_app.py](#))

Características:

- ⌚ Design moderno com gradientes CSS, cards, badges de status
- 💻 Layout responsivo com sidebar configurável
- 💾 Suporte a upload de XML e PDF em tabs separadas
- 📊 Métricas visuais (valor total, impostos, CFOP, natureza de operação)
- {EIF} Sistema de abas: Visão Geral, Partes (emitente/destinatário), Itens Detalhados, Impostos, Dados Técnicos
- ☑️ Formatação brasileira automática (CNPJ, CPF, CEP, telefone, valores monetários)
- 🔍 Revisão humana integrada com formulário completo
- 📥 Downloads em JSON (resultado completo e classificação isolada)

Fluxo de Uso:

- 1. Configuração:** Usuário informa URL do backend na sidebar
- 2. Upload:** Arrasta XML ou PDF para área de upload
- 3. Análise:** Clique em "Analizar com IA" dispara chamada à API
- 4. Resultado:** Sistema exibe métricas, abas com detalhes, classificação contábil
- 5. Revisão (se necessário):** Formulário aparece pedindo classificação manual
- 6. Aprendizado:** Revisão é persistida no CSV e workflow reprocessa

Tela Principal - Resumo da NF-e:

| Valor Total | Total Impostos | CFOP | Natureza |
|--------------|----------------|------|--------------------------|
| R\$ 1.500,00 | R\$ 234,56 | 5102 | Interestadual SP → RJ |

Aba Visão Geral - Classificação Contábil:

| Conta Débito: Clientes | Natureza: Interestadual |
|------------------------|-------------------------|
| Conta Crédito: Receita | Confiança: 90.0% |

Justificativa:
 Venda de mercadoria adquirida ou recebida de terceiros.
 Natureza: interestadual. Valor total da NF-e considerado para contexto: 1500.00.

Aba Partes - Emitente e Destinatário:

- Expanders com dados completos
- CNPJ/CPF formatados (12.345.678/0001-95)
- Endereço completo formatado
- Telefone formatado ((11) 98765-4321)

Aba Itens Detalhados:

- Lista com expanders por item
- NCM, CEST, código do produto
- Quantidade, unidade comercial, valor unitário
- Validação cruzada de cálculo (qtd × valor unitário ≈ valor)

Aba Impostos:

- Totais consolidados (ICMS, IPI, PIS, COFINS)
- Detalhamento por item quando disponível
- CST/CSOSN, origem, base de cálculo, alíquota, valor

Aba Dados Técnicos:

- JSON completo do payload
- JSON da classificação
- Botões de download (resultado completo, classificação isolada)

5.3 Formatadores Brasileiros ([src/utils/formatters.py](#))

```
def format_cnpj(cnpj: str) -> str:  
    """12345678000195 → 12.345.678/0001-95"""  
  
def format_cpf(cpf: str) -> str:  
    """12345678901 → 123.456.789-01"""  
  
def format_cep(cep: str) -> str:  
    """01310100 → 01310-100"""  
  
def format_telefone(telefone: str) -> str:  
    """11987654321 → (11) 98765-4321"""  
  
def format_valor_monetario(valor: float) -> str:  
    """1234.56 → R$ 1.234,56"""  
  
def format_quantidade(qtd: float) -> str:  
    """10.5 → 10,5000"""  
  
def format_endereco_completo(entidade: Union[Emitente, Destinatario]) -> str:  
    """Retorna endereço formatado: "Rua das Flores, 123 - Centro - São Paulo/SP -  
    CEP: 01310-100" """
```

6. Testes e Casos de Uso

6.1 Suite de Testes ([tests/](#))

Testes Implementados:

- [test_xml_parser.py](#): Parsing de XML com múltiplos cenários (simples nacional, CEST, impostos)
- [test_pdf_parser.py](#): Extração de PDF via LLM (requer chaves de API)
- [test_pdf_ie_destinatario.py](#): Validação de IE do destinatário (não confundir com emitente)
- [test_csosn_cest.py](#): Validação de CSOSN (Simples Nacional) e CEST (Substituição Tributária)

Executar Testes:

```
# Todos os testes
uv run -m pytest -q

# Testes específicos
uv run -m pytest tests/test_xml_parser.py -v

# Com cobertura
uv run -m pytest --cov=src --cov-report=html
```

6.2 Exemplos de Dados

XMLs de Exemplo ([data/exemplos/xml/](#)):

- [nfe_exemplo_1.xml](#): NF-e básica com todos campos obrigatórios
- [nfe_exemplo_2.xml](#): NF-e com múltiplos itens e impostos detalhados
- [nfe_exemplo_3.xml](#): NF-e com destinatário pessoa física (CPF)
- [nfe_simples_nacional.xml](#): NF-e com CSOSN (Simples Nacional)
- [nfe_com_cest.xml](#): NF-e com código CEST (Substituição Tributária)
- [nota_minima.xml](#): NF-e mínima para testes rápidos

PDFs de Exemplo ([data/exemplos/pdf/](#)):

- 5 exemplos de DANFE com layouts variados
- Testam diferentes provedores de LLM
- Validam OCR automático quando necessário

6.3 Casos de Uso Principais

Caso 1: Processamento de NF-e XML (Fluxo Completo Automático)

Entrada: XML de venda interestadual (SP→RJ), CFOP 5102, 3 itens, valor R\$ 1.500,00 **Processamento:**

1. XmlParserAgent extrai dados completos (emitente, destinatário, itens, impostos)
2. Validação Pydantic garante conformidade total
3. ClassificadorContabilAgent encontra CFOP 5102 no CSV
4. Determina natureza interestadual (SP≠RJ)

5. Retorna confiança 0.90 (alta), não requer revisão **Saída:** Classificação automática (Débito: Clientes, Crédito: Receita de Vendas)

Caso 2: Processamento de PDF com LLM (OpenAI)

Entrada: PDF do DANFE sem camada de texto (escaneado) **Processamento:**

1. PyMuPDF detecta ausência de texto
2. OCR automático via Tesseract extrai texto
3. Texto enviado ao GPT-4o-mini com schema JSON detalhado
4. LLM extrai 14 campos (emitente, destinatário, itens, impostos)
5. Sanitização normaliza CNPJ, UF, NCM, valores monetários
6. Validação Pydantic garante conformidade **Saída:** NFePayload idêntico ao que seria extraído de XML

Caso 3: Revisão Humana (CFOP Não Mapeado)

Entrada: NF-e com CFOP 6949 (operação especial não cadastrada) **Processamento:**

1. Parsing bem-sucedido via XML ou PDF
2. ClassificadorContabilAgent não encontra 6949 no CSV
3. Aplica fallback por prefixo (6xxx = saída)
4. Confiança 0.50 (baixa) → `needs_human_review=True`
5. Workflow seta `human_review_pending=True` e finaliza **Interação Humana:**
6. UI Streamlit exibe formulário de revisão
7. Usuário informa contas corretas, regime, justificativa, confiança
8. Frontend reenvia para `/classificar/review/xml` com input humano **Reprocessamento:**
9. Workflow aplica classificação manual
10. `upsert_cfop_mapping()` persiste no CSV
11. Próximas NF-es com CFOP 6949 serão automáticas **Saída:** Classificação corrigida + aprendizado persistido

Caso 4: Integração via API (Sistema Externo)

Cenário: ERP envia NF-e via API REST para classificação automática **Fluxo:**

```
# 1. Upload via API
curl -X POST "http://api.empresacom/classificar/xml" \
-F "xml_file=@nfe_12345.xml"

# 2. Resposta JSON
{
  "ok": true,
  "payload": {...},
  "classificacao": {
    "conta_debito": "1.1.2.01.0001",
    "conta_credito": "3.1.1.01.0002",
    "confianca": 0.95
  },
  "human_review_pending": false
}
```

```
}
```

```
# 3. ERP persiste classificação automaticamente
```

7. Configuração e Execução do Ambiente

7.1 Pré-requisitos do Sistema

🐍 Python 3.10+

- **Verificação:** `python --version`
- **Windows:** Instalar via [python.org](https://www.python.org/)
- **Linux/Mac:** `pyenv install 3.10` ou gerenciador de pacotes

⚡ UV (Gerenciador de Dependências)

- **Instalação Windows:** `powershell -c "irm https://astral.sh/uv/install.ps1 | iex"`
- **Instalação Linux/Mac:** `curl -LsSf https://astral.sh/uv/install.sh | sh`
- **Verificação:** `uv --version`

🔍 Tesseract OCR (Opcional, para PDFs escaneados)

- **Windows:** Baixar instalador de [tesseract-ocr](#)
- **Linux:** `sudo apt install tesseract-ocr tesseract-ocr-por`
- **Mac:** `brew install tesseract tesseract-lang`
- **Verificação:** `tesseract --version`

7.2 Setup Completo do Projeto

🛠️ Preparação do Ambiente

```
# Clone do repositório
git clone https://github.com/seu-usuario/agentes_contabeis.git
cd agentes_contabeis

# Criar ambiente virtual com UV
uv venv .venv

# Ativar ambiente
./.venv/Scripts/Activate.ps1 # Windows PowerShell
source .venv/bin/activate     # Linux/Mac

# Sincronizar dependências
uv pip sync requirements.txt
```

📁 Estrutura de Diretórios

```
agentes_contabeis/
|__ src/
    |__ agents/           # Agentes especializados
    |__ api/              # FastAPI endpoints
```

```

    └── app/                      # CLIs e UI Streamlit
    └── domain/                  # Modelos Pydantic
    └── workflow/                # LangGraph (graph, nodes, state)
    └── utils/                   # Formatadores e helpers
    └── data/
        └── exemplos/
            ├── xml/             # 7 XMLs de exemplo
            └── pdf/              # 5 PDFs de exemplo
    └── data_sources/
        └── contas_por_cfop.csv # Mapa CFOP → contas
    └── tests/                    # Suite de testes pytest
    └── logs/                     # Logs estruturados (gerados automaticamente)
    └── requirements.txt          # Dependências Python
    └── run.bat                  # Automação Windows
    └── .env                      # Configurações (criar manualmente)

```

7.3 Configuração de Variáveis de Ambiente

Crie arquivo `.env` na raiz do projeto:

```

# LLM Provider (openai | gemini | groq)
PDF_LLM_PROVIDER=openai

# Chaves de API (configurar conforme provedor escolhido)
OPENAI_API_KEY=sk-proj-...
GOOGLE_API_KEY=AIza...
GROQ_API_KEY=gsk_...

# Modelo (opcional, usa defaults se não especificado)
# openai: gpt-4o-mini | gpt-4o
# gemini: gemini-1.5-pro | gemini-1.5-flash
# groq: llama-3.1-70b-versatile | mixtral-8x7b-32768
PDF_LLM_MODEL=gpt-4o-mini

# Temperatura (0.0 = determinístico, 1.0 = criativo)
PDF_LLM_TEMPERATURE=0.0

```

7.4 Execução e Monitoramento

► Comandos de Execução

Testes:

```

# Todos os testes
uv run -m pytest -q

# Testes com output detalhado
uv run -m pytest -v

```

```
# Testes com cobertura
uv run -m pytest --cov=src --cov-report=html
```

CLIs:

```
# Parser simples (XML)
uv run -m src.app.parse_cli --xml data/exemplos/xml/nfe_exemplo_1.xml

# Workflow completo (XML)
uv run -m src.app.run_graph --xml data/exemplos/xml/nfe_exemplo_1.xml --regime
simples

# Workflow completo (PDF)
uv run -m src.app.run_graph --pdf data/exemplos/pdf/nfe_exemplo_1.pdf --regime
simples

# Com revisão humana
uv run -m src.app.run_graph --xml data/exemplos/xml/nfe_exemplo_1.xml --human-
review-json revisao.json
```

API REST:

```
# Iniciar servidor FastAPI
uv run uvicorn src.api.main:app --reload

# Testar healthcheck
curl http://localhost:8000/health

# Docs interativas
# Abrir http://localhost:8000/docs no navegador
```

Interface Web:

```
# Iniciar Streamlit
uv run streamlit run src/app/streamlit_app.py

# Interface abrirá automaticamente em http://localhost:8501
```

Automação Windows:

```
# Processa todos XMLs e PDFs de data/exemplos/ recursivamente
./run.bat

# Gera logs/ com:
# - Arquivos individuais .log para cada documento
```

```
# - summary.csv com resultado geral (ok, needs_review, reason)
# - Loop interativo para revisão humana quando necessário
```

📊 Interpretação de Resultados

Output CLI:

```
[INFO] NFe parse OK | cfop=5102 emitente=EMPRESA XYZ LTDA (CNPJ: 12345678000195,
UF: SP)
          destinatario=CLIENTE ABC (CNPJ: 98765432000156, UF: RJ) itens=3
vtotal=1500.00
[INFO] Classificação OK | cfop=5102 natureza=interestadual conta_debito=Clientes
      conta_credito=Receita de Vendas confianca=0.90 regime_tributario=simples
      fonte=csv needs_human_review=False
```

Output API (JSON):

```
{
  "ok": true,
  "payload": {
    "cfop": "5102",
    "emitente": {...},
    "destinatario": {...},
    "valor_total": 1500.00,
    "itens": [...]
  },
  "classificacao_ok": true,
  "classificacao": {
    "cfop": "5102",
    "natureza_operacao": "interestadual",
    "conta_debito": "Clientes",
    "conta_credito": "Receita de Vendas",
    "justificativa": "...",
    "confianca": 0.90,
    "needs_human_review": false
  },
  "human_review_pending": false
}
```

Logs Estruturados ([logs/](#)):

```
2025-01-30 14:23:45,123 [INFO] src.agents.xml_parser_agent: NFe parse OK |
cfop=5102 ...
2025-01-30 14:23:45,234 [INFO] src.agents.classificador_contabil_agent: Mapa CFOP
carregado: 25 linhas
2025-01-30 14:23:45,345 [INFO] src.agents.classificador_contabil_agent:
Classificação OK | ...
```


Conclusões Práticas

1. Automação e Eficiência Operacional

O Sistema de Extração de Dados Fiscais **elimina processamento manual** de documentos fiscais:

- **De:** Horas de digitação e classificação manual por contador
- **Para:** Segundos de processamento automatizado via IA
- **Inclui:** Parsing, validação, classificação contábil e persistência de aprendizado

2. Precisão e Confiabilidade

Arquitetura multiagente com **validação em múltiplas camadas** garante:

- **Parsing robusto** com dupla tentativa (XML) e multi-provedor (PDF)
- **Validação Pydantic rigorosa** com field validators e model validators
- **Classificação contábil baseada em regras** com scoring de confiança
- **Human-in-the-loop integrado** para casos de baixa confiança
- **Rastreabilidade completa** com logs estruturados e estado compartilhado

3. Escalabilidade Tecnológica

LangGraph e arquitetura modular permitem:

-  **Adição fácil de novos provedores de LLM** (basta implementar cliente LangChain)
-  **Modificação de regras de classificação** via CSV sem alterar código
-  **Processamento em lote** via `run.bat` ou integração com schedulers
-  **Integração com ERPs** via API REST bem documentada (OpenAPI/Swagger)

4. Conformidade Fiscal Brasileira

Implementação automática de:

-  **Validações fiscais específicas:** CFOP (4 dígitos), NCM (8 dígitos), CEST (7 dígitos)
-  **Impostos brasileiros:** ICMS (CST/CSOSN), IPI, PIS, COFINS
-  **Formato SEFAZ:** Parsing direto de XMLs com namespace handling
-  **Classificação contábil:** Débito/Crédito por CFOP e regime tributário

5. Manutenibilidade e DX Superior

-  **Separação clara** de responsabilidades (agents, models, workflow, api, app)
-  **Testes unitários** com pytest e cobertura de código
-  **Atualizações facilitadas** via UV (sync de dependências em segundos)
-  **Integração simplificada** com FastAPI async e Pydantic
-  **Deploy simplificado** (requirements.txt + .env + UV)

6. Observabilidade Completa

-  **Logs estruturados** com timestamps, níveis e contexto por agente
-  **Estado compartilhado** inspecionável via LangGraph
-  **Rastreabilidade de fluxo** (cada transição registrada)

- **Facilita auditoria** com logs persistidos e summary.csv
- **Otimização contínua** com scoring de confiança e aprendizado humano

7. Diferenciais Competitivos

- **Multi-formato nativo**: XML (parsing direto) e PDF (LLM)
 - **Multi-provedor de IA**: OpenAI, Gemini, Groq (configurável via .env)
 - **Aprendizado contínuo**: Revisões humanas persistidas no CSV
 - **UI moderna**: Streamlit com design profissional e UX intuitiva
 - **Performance**: UV (10-100x mais rápido que pip), LangGraph (execução determinística)
-

Relatório compilado em Outubro de 2025

Versão 1.0

Sistema de Extração de Dados Fiscais - Arquitetura Multiagente com IA