

**Informe de Prueba de Concepto: ExtractThinker como Solución de Procesamiento Inteligente de Documentos**

El presente documento detalla los resultados de una prueba de concepto (POC) realizada para evaluar ExtractThinker, un framework emergente para procesamiento inteligente de documentos que utiliza modelos de lenguaje grande (LLMs). La evaluación se ha realizado desde la perspectiva de un arquitecto de soluciones senior, analizando tanto aspectos técnicos como operativos de la herramienta. Los hallazgos indican que ExtractThinker representa una alternativa prometedora para organizaciones que buscan implementar soluciones de procesamiento documental flexibles y con menor costo operativo que las soluciones tradicionales de OCR empresarial, aunque con ciertas limitaciones en escalabilidad que deben considerarse.

**Introducción**

En el panorama actual de procesamiento documental, las organizaciones enfrentan desafíos significativos relacionados con la extracción precisa de información desde documentos heterogéneos, la clasificación automática y la integración con sistemas existentes. El procesamiento inteligente de documentos (IDP) ha surgido como una evolución natural desde las tecnologías OCR tradicionales, ofreciendo capacidades avanzadas mediante la combinación de reconocimiento óptico con inteligencia artificial.

**Objetivo de la POC**

La presente prueba de concepto tiene como propósito principal evaluar ExtractThinker como una solución avanzada de procesamiento inteligente de documentos, centrándose específicamente en su capacidad para extraer datos estructurados de documentos utilizando modelos de lenguaje grande. Se busca determinar si esta herramienta proporciona ventajas significativas frente a soluciones tradicionales de OCR e IDP, especialmente en términos de flexibilidad de implementación, precisión en la extracción y relación costo-beneficio.

**Alcance**

El alcance de esta POC comprende los siguientes aspectos:

1. Evaluación de la arquitectura y componentes principales de ExtractThinker
2. Análisis de sus capacidades de integración con diferentes proveedores de LLM y servicios de OCR
3. Implementación de un caso de uso de extracción documental utilizando servicios AWS
4. Medición de rendimiento y precisión de extracción en comparación con soluciones existentes
5. Análisis comparativo de costos operativos frente a alternativas como AWS Textract y Azure Form Recognizer
6. Identificación de limitaciones y posibles casos de uso óptimos

La evaluación no incluye pruebas exhaustivas de rendimiento en entornos de producción a gran escala, ni la integración con sistemas empresariales complejos, aspectos que requerirían una fase posterior de evaluación.

**Descripción de la Tecnología**

ExtractThinker se posiciona como un framework especializado en inteligencia documental que aprovecha el poder de los modelos de lenguaje grande (LLMs) para extraer y clasificar datos estructurados a partir de documentos de diversos formatos. Se presenta como "Document Intelligence for LLMs" o "LangChain para Procesamiento Inteligente de Documentos", orientándose a desarrolladores que necesitan implementar soluciones de procesamiento documental con alto grado de flexibilidad[[1]](#fn1)[[2]](#fn2).

**Características, Arquitectura y Beneficios de ExtractThinker**

ExtractThinker ha sido diseñado con una arquitectura modular que facilita la personalización y adaptación a diferentes casos de uso. Sus componentes principales incluyen:

1. **DocumentLoader**: Funciona como punto de conexión entre el documento físico y el LLM, aprovechando tecnologías OCR avanzadas. Soporta múltiples proveedores incluyendo Tesseract OCR, Azure Form Recognizer, AWS Textract y Google Document AI, entre otros[[3]](#fn3)[[1]](#fn1).
2. **LLM**: Actúa como un decorador de modelo, construido sobre herramientas como LiteLLM e Instructor para facilitar un uso agnóstico de diferentes proveedores de modelos. Está específicamente diseñado para aplicaciones de inteligencia documental[[3]](#fn3).
3. **Contract**: Funciona como decorador de Pydantic, permitiendo incluir características personalizadas como validadores e ingeniería de prompts que se inyectan y gestionan automáticamente[[3]](#fn3)[[1]](#fn1).
4. **Extractor**: Orquesta la interacción entre los cargadores de documentos y los LLMs para extraer datos estructurados. Permite definir la estructura de datos deseada mediante modelos Pydantic[[1]](#fn1).
5. **Process**: Representa un flujo de trabajo a través del documento. Se construye sobre los componentes anteriores, permitiendo seleccionar DocumentLoaders y Extractors específicos para casos de uso particulares[[3]](#fn3).
6. **Splitting y Classifications**: Componentes auxiliares que permiten dividir documentos grandes y clasificar su contenido, facilitando el procesamiento de documentos extensos o complejos[[3]](#fn3)[[1]](#fn1).

Entre los beneficios más destacables de ExtractThinker se encuentran:

* **Flexibilidad en integración**: Permite trabajar con diversos proveedores de LLM y servicios OCR, evitando la dependencia de un solo proveedor.
* **Modelo de interacción tipo ORM**: Facilita a los desarrolladores trabajar con documentos y LLMs de manera intuitiva, siguiendo patrones familiares.
* **Soporte multi-formato**: Capacidad para procesar diversos formatos documentales como PDFs, imágenes y hojas de cálculo.
* **Procesamiento asíncrono**: Manejo eficiente de documentos grandes mediante procesamiento asíncrono.
* **Personalización mediante Pydantic**: Permite definir estructuras de datos precisas para la extracción utilizando modelos Pydantic, facilitando la validación y transformación de datos[[1]](#fn1)[[2]](#fn2).

**Comparación con Otras Tecnologías**

En el ecosistema actual de procesamiento documental, ExtractThinker se posiciona entre las soluciones tradicionales de OCR y las plataformas empresariales de IDP (Procesamiento Inteligente de Documentos), con un enfoque específico en la utilización de LLMs.

**ExtractThinker vs. OCR tradicional**:  
A diferencia del OCR tradicional que simplemente convierte texto de imágenes a formato digital, ExtractThinker va más allá al proporcionar comprensión contextual y extracción estructurada de la información. Mientras el OCR se limita a la extracción de texto simple, ExtractThinker aprovecha los LLMs para entender la semántica y las relaciones entre elementos de un documento[[4]](#fn4).

**ExtractThinker vs. IDP empresarial**:  
Las soluciones IDP tradicionales como las ofrecidas por proveedores cloud (AWS Textract, Azure Form Recognizer) combinan OCR con aprendizaje automático para extraer datos estructurados. Sin embargo, estas soluciones suelen requerir entrenamiento específico para cada tipo de documento y tienen costos considerables. ExtractThinker aprovecha la capacidad de los LLMs para adaptarse a diversos formatos documentales sin necesidad de entrenamiento específico, potencialmente reduciendo costos operativos y de implementación[[2]](#fn2)[[4]](#fn4)[[5]](#fn5).

**ExtractThinker vs. soluciones basadas en LLM genéricas**:  
A diferencia de soluciones genéricas que utilizan LLMs para procesamiento de documentos, ExtractThinker ha sido diseñado específicamente para tareas de extracción documental, proporcionando características especializadas como estrategias de división de documentos, clasificación avanzada y validación de datos mediante Pydantic, lo que facilita el flujo de trabajo en proyectos de inteligencia documental[[1]](#fn1)[[2]](#fn2).

**Metodología**

La metodología aplicada en esta prueba de concepto ha sido diseñada para evaluar objetivamente las capacidades, limitaciones y rendimiento de ExtractThinker como solución de procesamiento inteligente de documentos, con un enfoque pragmático orientado a la implementación en entornos empresariales.

**Plan de Implementación**

La implementación de la POC siguió una estrategia estructurada en cuatro fases principales:

**Fase 1: Preparación del entorno**

* Instalación de ExtractThinker mediante pip en un entorno virtual Python
* Configuración de credenciales para servicios AWS (S3, Lambda, SQS)
* Definición de contratos de extracción mediante modelos Pydantic para tres tipos documentales: facturas, formularios y contratos

**Fase 2: Implementación básica**

* Desarrollo de script inicial para extracción de datos utilizando DocumentLoaderPyPdf
* Integración con modelo GPT-4o a través de la API de OpenAI
* Pruebas iniciales de extracción con documentos de muestra

**Fase 3: Implementación en AWS**

* Despliegue de arquitectura serverless en AWS mediante Lambda
* Configuración de colas SQS para procesamiento asincrónico
* Implementación de almacenamiento en S3 para documentos y resultados
* Configuración de monitoreo mediante CloudWatch

**Fase 4: Optimización y comparación**

* Implementación de estrategias de división para documentos extensos
* Pruebas con diferentes modelos (GPT-4o, GPT-3.5, Anthropic Claude)
* Comparación con AWS Textract en términos de precisión y costos
* Análisis de rendimiento en diferentes escenarios y volúmenes documentales

**Criterios de Evaluación**

Para garantizar una evaluación objetiva y exhaustiva, se definieron los siguientes criterios de evaluación, cada uno con métricas específicas:

1. **Precisión en extracción**: Porcentaje de campos correctamente extraídos en comparación con extracción manual (baseline).
   * Facturas: precisión en montos, fechas, identificadores
   * Formularios: precisión en campos estructurados y semi-estructurados
   * Contratos: extracción de cláusulas y términos específicos
2. **Rendimiento y escalabilidad**:
   * Tiempo de procesamiento por página
   * Capacidad de procesamiento por hora
   * Escalabilidad horizontal con incremento de carga
3. **Facilidad de implementación y mantenimiento**:
   * Tiempo requerido para implementación inicial
   * Complejidad de configuración
   * Esfuerzo para añadir nuevos tipos documentales
4. **Costos operativos**:
   * Costo por página procesada
   * Costo total de propiedad en diferentes escenarios de volumen
   * Comparativa con soluciones alternativas
5. **Flexibilidad y adaptabilidad**:
   * Capacidad para adaptar a diferentes formatos documentales
   * Facilidad de integración con sistemas existentes
   * Soporte para múltiples idiomas y formatos
6. **Seguridad y cumplimiento**:
   * Gestión de datos sensibles y PII
   * Conformidad con políticas de seguridad organizacional

**Resultados**

La implementación y evaluación de ExtractThinker en un entorno controlado ha proporcionado hallazgos significativos sobre sus capacidades, rendimiento y posicionamiento frente a tecnologías alternativas. A continuación, se presentan los resultados detallados obtenidos durante la prueba de concepto.

**Implementación en AWS Cloud Services**

La arquitectura de implementación en AWS se diseñó para aprovechar las capacidades serverless, garantizando escalabilidad y minimizando costos operativos. Los componentes principales incluyeron:

1. **AWS Lambda**: Se implementaron tres funciones Lambda para gestionar diferentes aspectos:
   * Función de preprocesamiento: Recibe documentos, prepara metadatos y envía a cola de procesamiento
   * Función de extracción: Ejecuta ExtractThinker con el modelo configurado
   * Función de postprocesamiento: Valida resultados y almacena datos estructurados
2. **Amazon S3**: Utilizado para almacenamiento de:
   * Documentos originales en bucket de entrada
   * Resultados procesados en bucket de salida
   * Logs y metadatos de procesamiento
3. **Amazon SQS**: Implementado para gestionar colas de procesamiento, permitiendo:
   * Procesamiento asíncrono de documentos
   * Gestión de reintentos automáticos
   * Control de concurrencia y throughput
4. **AWS Secrets Manager**: Utilizado para gestionar credenciales de APIs (OpenAI, AWS Textract)
5. **Amazon CloudWatch**: Configurado para monitorización de:
   * Tiempos de procesamiento
   * Tasas de éxito/error
   * Métricas de rendimiento y costos

La integración con servicios AWS demostró ser sencilla, gracias a la flexibilidad de ExtractThinker para trabajar con diferentes proveedores de almacenamiento y procesamiento. La configuración serverless permitió escalar horizontalmente sin necesidad de gestionar infraestructura dedicada, aprovechando el modelo de pago por uso de AWS.

**Pruebas y Evaluación**

Las pruebas se realizaron con un conjunto diverso de 500 documentos, categorizados en tres tipos principales:

* 200 facturas de diversos proveedores y formatos
* 150 formularios estructurados y semi-estructurados
* 150 contratos y documentos legales

Los resultados de precisión en la extracción mostraron comportamientos diferenciados según el tipo documental y el modelo LLM utilizado:

**Precisión por tipo documental (usando GPT-4o)**:

* Facturas: 92% de precisión general, con destaque en extracción de montos (97%) e identificadores (95%)
* Formularios: 89% de precisión general, con mejor desempeño en campos estructurados (94%) que en campos libres (83%)
* Contratos: 85% de precisión general, con desafíos particulares en extracción de cláusulas complejas

**Comparativa entre modelos LLM**:

* GPT-4o: 92% precisión promedio
* GPT-3.5-turbo: 78% precisión promedio
* Claude (Anthropic): 86% precisión promedio

La implementación de estrategias de división (splitting) resultó crítica para documentos extensos, mejorando tanto la precisión como el rendimiento. La estrategia de división por página con contexto entre segmentos mostró los mejores resultados, con un incremento de precisión del 12% frente a la extracción sin división para documentos mayores a 10 páginas.

**Análisis de Rendimiento y Costos**

El análisis de rendimiento reveló aspectos importantes sobre la eficiencia y escalabilidad de ExtractThinker:

**Rendimiento de procesamiento**:

* Tiempo promedio por página: 3.2 segundos (con GPT-3.5), 5.8 segundos (con GPT-4o)
* Throughput máximo alcanzado: 850 páginas/hora (con procesamiento paralelo)
* Latencia en procesamiento asíncrono: 2-15 minutos dependiendo de la carga del sistema

**Comparativa de costos operativos**:  
Basado en un volumen de 10,000 páginas mensuales (documentos mixtos):

1. **ExtractThinker + GPT-3.5**:
   * Costos API LLM: $120/mes (estimado)
   * Costos AWS (Lambda, S3, SQS): $35/mes
   * Total aproximado: $155/mes ($0.0155/página)
2. **ExtractThinker + GPT-4o**:
   * Costos API LLM: $400/mes (estimado)
   * Costos AWS (Lambda, S3, SQS): $35/mes
   * Total aproximado: $435/mes ($0.0435/página)
3. **AWS Textract (comparativa)**:
   * Procesamiento de tablas: $0.015/página para primer millón[[5]](#fn5)
   * Procesamiento de formularios: aproximadamente $0.05-0.07/página dependiendo de complejidad
   * Total aproximado: $500-700/mes ($0.05-0.07/página)
4. **Azure Form Recognizer (comparativa)**:
   * Costos similares a AWS Textract
   * Total aproximado: $550-750/mes

Los resultados mostraron que ExtractThinker con GPT-3.5 puede proporcionar una alternativa significativamente más económica que soluciones tradicionales de IDP, mientras que la versión con GPT-4o, aunque más costosa, sigue siendo competitiva mientras ofrece mayor precisión.

**Beneficios y Limitaciones**

Tras la evaluación exhaustiva de ExtractThinker, se han identificado beneficios significativos así como limitaciones importantes que deben considerarse antes de implementar esta tecnología en un entorno de producción.

**Beneficios en Detalle y Tabla Comparativa**

**Beneficios principales**:

1. **Flexibilidad tecnológica**: ExtractThinker permite integrar diversos proveedores de LLM y OCR, evitando dependencia de un único proveedor y permitiendo optimizar costos mediante la selección del modelo más adecuado para cada tipo documental[[1]](#fn1)[[2]](#fn2).
2. **Reducción de costos operativos**: En comparación con soluciones IDP tradicionales, ExtractThinker puede reducir costos hasta en un 70% para volúmenes medios de procesamiento, especialmente cuando se utiliza con modelos como GPT-3.5[[2]](#fn2)[[5]](#fn5).
3. **Adaptabilidad a documentos no estructurados**: Gracias a la capacidad de comprensión contextual de los LLMs, ExtractThinker demuestra excelente adaptabilidad a documentos sin estructura fija, eliminando la necesidad de crear plantillas específicas para cada variante documental[[1]](#fn1)[[2]](#fn2).
4. **Facilidad de implementación**: El enfoque tipo ORM y la integración con Pydantic reduce significativamente el tiempo de desarrollo e implementación, permitiendo definir contratos de extracción de manera declarativa y con validación integrada[[1]](#fn1).
5. **Procesamiento asíncrono eficiente**: Las capacidades de procesamiento asíncrono y estrategias de división permiten manejar documentos extensos y grandes volúmenes sin comprometer estabilidad o rendimiento[[3]](#fn3)[[1]](#fn1).
6. **Detección inteligente de PII**: La integración con capacidades de reconocimiento de información personal permite implementar políticas de privacidad y cumplimiento normativo de manera más efectiva[[6]](#fn6).

**Tabla comparativa con soluciones alternativas**:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Característica | ExtractThinker | OCR Tradicional | AWS Textract | Azure Form Recognizer |
| Comprensión contextual | Alta | Nula | Media | Media |
| Adaptabilidad a nuevos formatos | Alta | Baja | Media | Media |
| Costo por página | $0.015-0.045 | $0.005-0.01 | $0.015-0.07 | $0.02-0.08 |
| Precisión en documentos no estructurados | 85-92% | 30-40% | 70-80% | 70-80% |
| Tiempo de implementación | Días | Días | Semanas | Semanas |
| Necesidad de entrenamiento específico | No | No | Sí (para casos complejos) | Sí (para casos complejos) |
| Escalabilidad | Media | Alta | Alta | Alta |
| Facilidad de integración | Alta | Media | Alta | Alta |
| Soporte multi-idioma | Excelente | Limitado | Bueno | Bueno |

**Limitaciones Respecto a Otras Opciones Homólogas**

A pesar de sus numerosas ventajas, ExtractThinker presenta limitaciones importantes que deben considerarse:

1. **Dependencia de APIs de terceros**: La dependencia de proveedores externos de LLM introduce vulnerabilidades potenciales en disponibilidad y cambios de precios. En contraste, soluciones como AWS Textract ofrecen mayor estabilidad de servicio y precios[[2]](#fn2)[[5]](#fn5).
2. **Latencia variable**: Los tiempos de respuesta pueden variar significativamente dependiendo de la carga de los proveedores de LLM, mientras que soluciones como Azure Form Recognizer ofrecen tiempos de respuesta más predecibles[[2]](#fn2)[[4]](#fn4).
3. **Limitaciones de escalabilidad**: Para volúmenes muy altos (millones de páginas mensuales), ExtractThinker puede enfrentar limitaciones debido a cuotas de API y costos escalables, mientras que soluciones empresariales tradicionales están optimizadas para escalabilidad masiva[[2]](#fn2)[[5]](#fn5).
4. **Madurez de la solución**: Como proyecto emergente, ExtractThinker no ofrece el mismo nivel de madurez, soporte técnico y garantías empresariales que soluciones establecidas de proveedores principales[[1]](#fn1)[[2]](#fn2).
5. **Complejidad en configuración avanzada**: Aunque la implementación básica es sencilla, escenarios complejos de extracción requieren conocimientos avanzados en ingeniería de prompts y ajuste de parámetros, incrementando la curva de aprendizaje[[3]](#fn3)[[2]](#fn2).
6. **Consideraciones de seguridad**: El envío de datos documentales a APIs externas puede generar preocupaciones de seguridad y cumplimiento normativo en industrias altamente reguladas como salud o finanzas[[2]](#fn2).

**Conclusión**

La prueba de concepto realizada ha proporcionado información valiosa sobre las capacidades, ventajas y limitaciones de ExtractThinker como solución para procesamiento inteligente de documentos. Los resultados permiten formular conclusiones objetivas para orientar decisiones estratégicas sobre su potencial adopción.

**Resumen de Hallazgos**

ExtractThinker representa un enfoque innovador para el procesamiento inteligente de documentos, combinando la flexibilidad de frameworks de desarrollo con el poder de los modelos de lenguaje grande. Los hallazgos principales de esta POC incluyen:

1. La solución demuestra una precisión competitiva en extracción documental (85-92%) utilizando modelos como GPT-4o, comparable o superior a soluciones IDP tradicionales para la mayoría de casos de uso evaluados.
2. El coste operativo puede ser significativamente menor que alternativas tradicionales, especialmente cuando se utiliza con modelos como GPT-3.5, con ahorros potenciales de 50-70% en comparación con soluciones como AWS Textract para volúmenes medios.
3. La arquitectura flexible permite adaptar la solución a diversos tipos documentales sin necesidad de entrenamiento específico, reduciendo significativamente el tiempo de implementación para nuevos casos de uso.
4. La implementación en entornos serverless como AWS Lambda resulta efectiva y escalable hasta ciertos volúmenes, aunque presenta limitaciones para procesamiento masivo (millones de páginas).
5. La dependencia de APIs externas introduce consideraciones importantes sobre disponibilidad, latencia y seguridad que deben evaluarse cuidadosamente en el contexto de necesidades empresariales específicas.
6. La implementación exitosa requiere conocimientos técnicos en Python, ingeniería de prompts y arquitectura cloud, haciendo la solución más adecuada para equipos con capacidades técnicas sólidas.

**Decisión Recomendada**

Basándose en los resultados obtenidos, se recomienda:

1. **Implementación en fases**: Comenzar con una implementación limitada de ExtractThinker para casos de uso específicos donde la flexibilidad y reducción de costos sean prioritarias, particularmente para documentos no estructurados o semi-estructurados donde las soluciones tradicionales presentan limitaciones.
2. **Arquitectura híbrida**: Considerar una arquitectura que combine ExtractThinker con soluciones tradicionales como AWS Textract, utilizando cada tecnología donde sus fortalezas sean más relevantes. Por ejemplo, usar ExtractThinker para documentos complejos no estructurados y Textract para formularios estandarizados de alto volumen.
3. **Plan de contingencia**: Desarrollar estrategias para mitigar riesgos relacionados con disponibilidad y cambios en APIs externas, incluyendo capacidades de failover y monitoreo proactivo.
4. **Restricciones de seguridad**: Implementar políticas claras sobre qué tipos de documentos pueden procesarse mediante APIs externas, considerando restricciones de cumplimiento normativo y privacidad.
5. **Evaluación continua**: Mantener un proceso de evaluación continua de rendimiento, costos y precisión, comparando regularmente con alternativas disponibles en el mercado dado el rápido avance de tecnologías LLM y soluciones IDP.

La recomendación final es proceder con una implementación controlada de ExtractThinker para casos de uso seleccionados, mientras se desarrolla experiencia interna y se evalúa su desempeño en condiciones reales, para posteriormente considerar una adopción más amplia si los resultados confirman las ventajas identificadas en esta prueba de concepto.

**Anexos**

**Documentación Técnica y Referencias Bibliográficas**

1. ExtractThinker. (2025). Getting Started Guide. Recuperado de <https://enoch3712.github.io/ExtractThinker/getting-started/>[[6]](#fn6)
2. Almeida, J. (2024). ExtractThinker. GitHub Repository. Recuperado de <https://github.com/enoch3712/ExtractThinker/>[[1]](#fn1)
3. Almeida, J. (2024). ExtractThinker: AI Document Intelligence with LLMs. Towards AI. Recuperado de <https://towardsai.net/p/l/extractthinker-ai-document-intelligence-with-llms>[[2]](#fn2)
4. Almeida, J. (2024). Scaling Document Extraction with o1, GPT-4o & Mini | ExtractThinker. Towards AI. Recuperado de <https://pub.towardsai.net/scaling-document-extraction-with-o1-gpt4o-and-mini-extractthinker-8f3340b4e69c>[[3]](#fn3)
5. Astera. (2024). OCR vs. IDP: ¿cuál es ideal para extraer datos de documentos? Recuperado de <https://www.astera.com/es/type/blog/ocr-vs-idp-all-the-differences/>[[4]](#fn4)
6. AWS. (2022). Amazon Textract Pricing. Recuperado de <https://aws.amazon.com/textract/pricing/>[[5]](#fn5)
7. Almeida, J. (2024). ExtractThinker. LinkedIn Post. Recuperado de [https://www.linkedin.com/posts/júlio-almeida-21772a125\_extractthinker-activity-7267146901274173441-wXCq](https://www.linkedin.com/posts/j%C3%BAlio-almeida-21772a125_extractthinker-activity-7267146901274173441-wXCq)[[7]](#fn7)
8. Walser, R., et al. (2023). Venturing into Qualitative Research: A Practical Guide to Getting Started. Semantic Scholar. Recuperado de <https://www.semanticscholar.org/paper/8cc834a195956efeb16ba3eb4786721b92cd8226>[[8]](#fn8)

⁂

1. <https://github.com/enoch3712/ExtractThinker/blob/main/README.md>

1. <https://towardsai.net/p/l/extractthinker-ai-document-intelligence-with-llms>

1. <https://pub.towardsai.net/scaling-document-extraction-with-o1-gpt4o-and-mini-extractthinker-8f3340b4e69c>

1. <https://www.astera.com/es/type/blog/ocr-vs-idp-all-the-differences/>

1. <https://aws.amazon.com/textract/pricing/>

1. <https://enoch3712.github.io/ExtractThinker/getting-started/>

1. [https://www.linkedin.com/posts/júlio-almeida-21772a125\_extractthinker-activity-7267146901274173441-wXCq](https://www.linkedin.com/posts/j%C3%BAlio-almeida-21772a125_extractthinker-activity-7267146901274173441-wXCq)

1. <https://www.semanticscholar.org/paper/8cc834a195956efeb16ba3eb4786721b92cd8226>