Boon AI Hackathon: TMS Conversion Solution

Transforming Shipping Documents into Standardized TMS Format

Initial Data: Understanding the Challenge

Source Data: Markdown Files

- **Format**: Unstructured text in markdown format
- **Content**: Shipping details, customer information, rates, pickup/delivery instructions
- **Challenges**: Inconsistent formatting, varied information structure
- **Example**:

```
 # Order Information
- Reference Number: 31421-79757
- Customer: Neon Logistics
- Equipment: Van

## Pickup Details
- Location: JAVA TRADING CO LLC
- Address: 815 Houser Way N, The Landing Renton, WA 98055
- Appointment: 12/31/24 12:00 - 15:30

## Delivery Details
- Location: Safeway - Salt Lake
- Address: 620 W 600 N North Salt Lake, UT 84054
- Appointment: 01/02/25 15:30
```

Target Data: TMS Format

- **Format**: Highly structured JSON with specific field requirements
- **Complexity**: Nested structures, IDs, specific formatting rules
- **Key Components**: Order details, stops, movements, reference numbers
- **Example Fields**:

```
{
  "__type": "orders",
  "company_id": "TMS",
  "blnum": "31421-79757",
  "customer_id": "NEONLOGI",
  "equipment_type_id": "V",
  "stops": [
    {
      "__type": "stop",
      "stop_type": "PU",
      "address": "815 HOUSER WAY NORTH",
      "city_name": "RENTON",
      "state": "WA"
    }
  ]
}
```

Step 1: Markdown to Unified JSON Conversion

Approach

- **LLM-Based Extraction**: Used OpenAI API to parse unstructured text
- **Prompt Engineering**: Designed prompts to extract specific fields
- **Standardization**: Created a unified JSON schema for consistent representation

Implementation

```
def extract_data_with_openai(markdown_content, api_key):
    client = OpenAI(api_key=api_key)
    response = client.chat.completions.create(
        model="gpt-4o-2024-11-20",
        messages=[
            {"role": "system", "content": EXTRACTION_SYSTEM_PROMPT},
            {"role": "user", "content": markdown_content}
        ],
        temperature=0.1
    )
    return parse_json_response(response.choices[0].message.content)
```

Unified JSON Format


```
{
  "equipment_type": "Van",
  "reference_number": "31421-79757",
  "total_rate": 2177.0,
  "freight_rate": 1250.0,
  "additional_rate": 927.0,
  "shipper_section": [
    {
      "ship_from_company": "JAVA TRADING CO LLC",
      "ship_from_address": "815 Houser Way N, The Landing Renton, WA 98055"
    }
  ],
  "receiver_section": [
    {
      "receiver_company": "Safeway - Salt Lake",
      "receiver_address": "620 W 600 N North Salt Lake, UT 84054"
    }
  ],
  "customer_name": "Neon Logistics"
}
```


Step 1 Results

- **Extraction Accuracy**: 83.7% overall field accuracy
- **Strengths**:
  - Reference numbers: 95.2% accuracy
  - Pickup/delivery dates: ~90% accuracy
  - Rate information: 88.3% accuracy
- **Challenges**:
  - Instruction fields: Only 20-30% accuracy
  - Multiple pickup/delivery locations: Lower presence rates
  - Equipment type standardization: 55.3% accuracy

Step 2: Unified JSON to TMS Format Conversion

Approach

- **Template-Based Conversion**: Created detailed TMS template for LLM
- **Customer ID Mapping**: Developed system to map customer names to TMS IDs
- **Field Transformation**: Implemented rules for formatting addresses, dates, etc.
- **Parallel Processing**: Optimized for performance with multi-threading

Implementation

```
def convert_with_llm(extraction_data):
    # Prepare the prompt with the extraction data and template
    prompt = f"{TMS_TEMPLATE}\nExtraction data:\n{json.dumps(extraction_data, indent=2)}"

    # Call OpenAI API
    response = client.chat.completions.create(
        model="gpt-4o-2024-11-20",
        messages=[
            {"role": "system", "content": "You convert extraction data to TMS format."},
            {"role": "user", "content": prompt}
        ],
        temperature=0.1
    )

    # Extract and process the response
    tms_json_str = response.choices[0].message.content
    # Process and return the TMS data
    return process_tms_response(tms_json_str)
```

TMS Format Output


```
{
  "__type": "orders",
  "company_id": "TMS",
  "blnum": "31421-79757",
  "customer_id": "NL",
  "equipment_type_id": "V",
  "freight_charge": 1250.0,
  "total_charge": 2177.0,
  "stops": [
    {
      "__type": "stop",
      "address": "815 Houser Way N",
      "city_name": "Renton",
      "state": "WA",
      "stop_type": "PU",
      "sched_arrive_early": "20241231120000-0700"
    },
    {
      "__type": "stop",
      "address": "620 W 600 N",
      "city_name": "North Salt Lake",
      "state": "UT",
      "stop_type": "SO",
      "sched_arrive_early": "20250102153000-0700"
    }
  ]
}
```


Performance Optimization

- **Sequential Processing**: ~21 seconds per file
- **Parallel Implementation**: ~2.93 seconds per file (7x improvement)
- **Total Processing Time**: 4 minutes 17 seconds for 88 files

```
# Parallel implementation with thread-local clients
def process_file(extraction_file, output_dir):
    # Thread-local OpenAI client
    client = get_thread_client()
    # Process file and convert to TMS format
    # ...

# Process files in parallel
with concurrent.futures.ThreadPoolExecutor(max_workers=10) as executor:
    futures = {executor.submit(process_file, file, output_dir): file for file in files}
    for future in tqdm(concurrent.futures.as_completed(futures), total=len(files)):
        # Process results
```

Step 2 Results & Challenges

Overall Performance

- **61.35% Average Accuracy** across 85 files

Strengths

- **Stop Types**: 100% accuracy
- **Equipment Type**: 95.12% accuracy
- **State Fields**: 94.19% accuracy
- **Reference Numbers**: 89.41% accuracy
- **Total Charges**: 89.02% accuracy

Key Challenges

1. Customer ID Mapping (2.35% accuracy)

- **Problem**: LLM generates logical abbreviations (e.g., "NL" for Neon Logistics) but ground truth uses different IDs (e.g., "NEONLOGI")
- **Example**:

  ```
  "customer_name": "Ryan Dougherty" → LLM: "RD" vs. Ground Truth: "ARMSCONC"
  ```

- **Solution**: Need comprehensive mapping from customer names to exact TMS IDs

2. Address Formatting (50.97% accuracy)

- **Problem**: Inconsistent formatting between LLM output and TMS requirements
- **Example**:

  ```
  LLM: "27255 SW 95TH AVE" vs. Ground Truth: "27255 SOUTH WEST 95TH AVENUE"
  ```

- **Solution**: Standardize address formatting with post-processing rules

3. Temperature Fields (0% accuracy)

- **Problem**: Complete failure to handle temperature requirements
- **Solution**: Add specific template instructions and examples

---

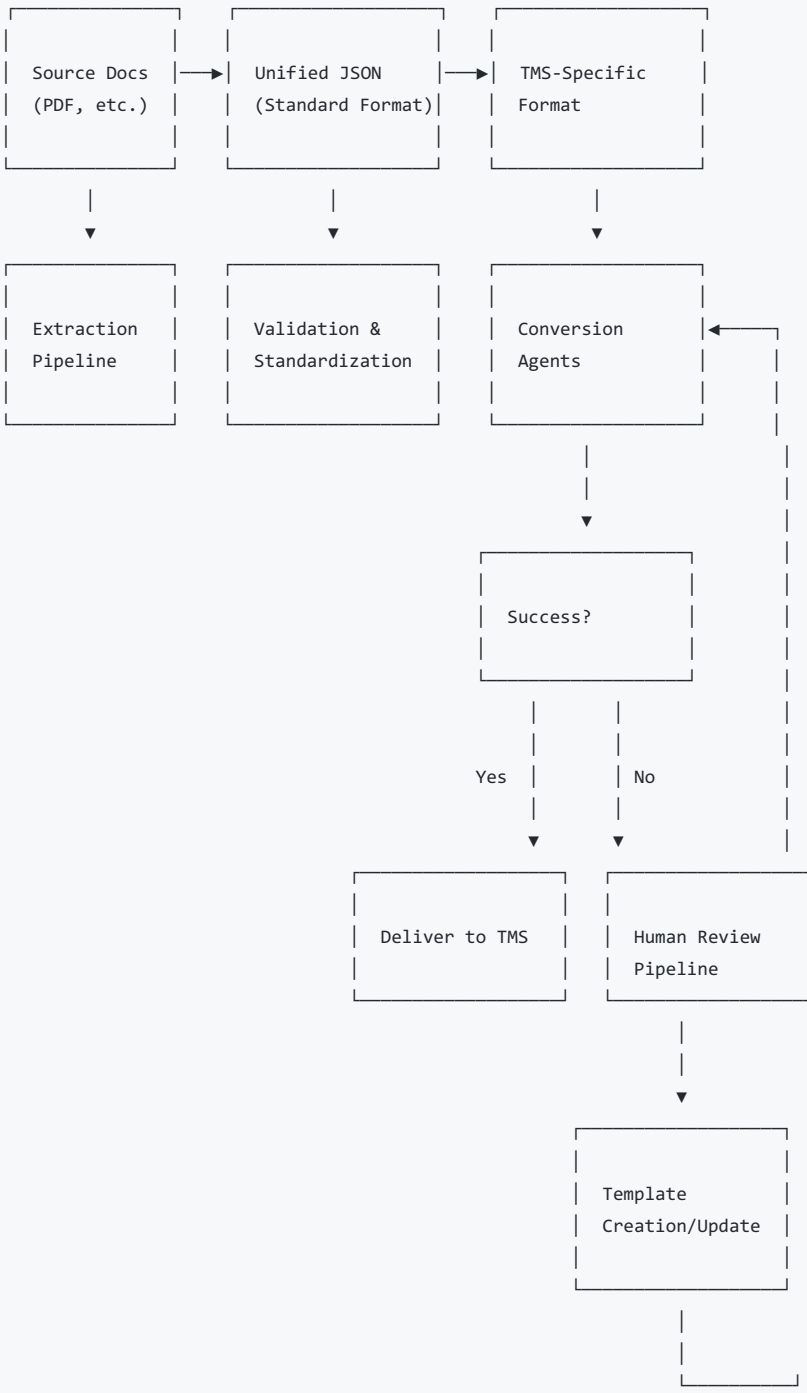
Field Accuracy Breakdown

Field	Accuracy
stop_type	100.00%
equipment_type_id	95.12%
state	94.19%
blnum	89.41%

Field	Accuracy
total_charge	89.02%
zip_code	88.39%
sched_arrive_early	78.06%
otherchargetotal	78.05%
freight_charge	73.17%
sched_arrive_late	63.23%
address	50.97%
city_name	45.81%
location_name	28.39%
customer_id	2.35%
temperature_min	0.00%
temperature_max	0.00%

Ideal Solution: Production Architecture

Enhanced Two-Step Approach



Lessons Learned

What Worked Well

1. **Two-Step Approach**: Separating extraction from conversion simplified the problem
2. **LLM-Based Conversion**: Handled complex transformations better than rule-based approaches
3. **Parallel Processing**: Significantly improved performance
4. **Field-Level Evaluation**: Provided detailed insights for targeted improvements

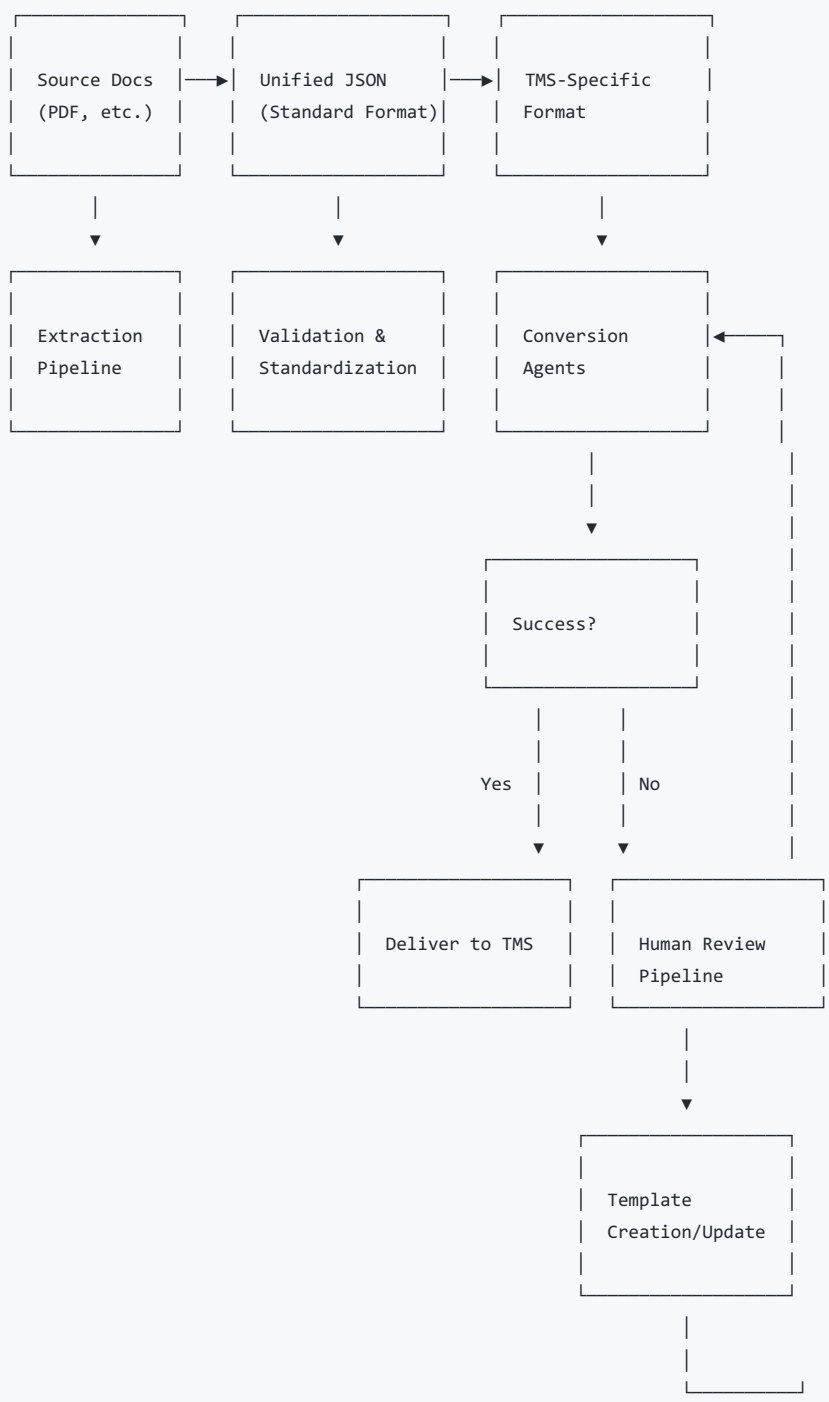
What Could Be Improved

1. **Customer ID Mapping**: Critical weakness that needs comprehensive solution
2. **Template Design**: More examples and specific formatting instructions needed
3. **Post-Processing**: Additional rules for standardization would improve accuracy
4. **Error Handling**: Better handling of edge cases and invalid data

---

Ideal Solution: Production Architecture

Enhanced Two-Step Approach



Key Components for Production

1. Unified JSON Schema Design

- Include all possible fields from various TMS systems
- Use clear, consistent naming conventions
- Support nested structures for complex data
- Include metadata about source document and confidence

2. TMS Format Registry

- Store format templates as JSON schemas or transformation rules
- Include version information for each TMS system
- Document field mappings and special handling requirements
- Track success rates to identify problematic conversions

3. Human Review Interface

- Simple UI for reviewing and correcting conversions
- Template editor for creating/updating TMS formats
- Dashboard for tracking conversion metrics
- Notification system for alerting reviewers

Recommendations for Immediate Improvements

1. Enhanced Customer ID Mapping

- Extract all customer IDs from ground truth TMS files
- Create comprehensive mapping file
- Implement fuzzy matching with higher thresholds

2. Improved Address Formatting

- Standardize address components
- Apply consistent casing (uppercase)
- Implement post-processing for abbreviations

3. Temperature Field Handling

- Add specific template instructions
- Include examples of correctly formatted fields
- Implement validation rules

4. Post-Processing Pipeline

- Add field-specific formatting rules
- Implement validation checks
- Create standardization functions for common fields

Benefits of Our Approach

1. **Scalability**: Add new document types and TMS formats without redesigning the system
2. **Quality**: Focus on accuracy at each specialized step
3. **Adaptability**: Respond to changes in TMS requirements quickly
4. **Learning**: Continuously improve through feedback loops
5. **Efficiency**: Minimize human intervention while maintaining high quality
6. **Transparency**: Clear visibility into conversion performance and issues

---

Next Steps

1. Implement enhanced customer ID mapping with ground truth data
2. Develop post-processing rules for address standardization
3. Create temperature field handling logic
4. Build a simple human review interface for edge cases
5. Expand the solution to handle additional document types (PDF, emails)

Thank You!

Questions?