ChatGPT

Recommendation Summary

Top Phase-A datasets: (1) **Freelance Job Postings (Upwork & Freelancer)** – large corpora of real job descriptions with budget fields [1]. These provide *title, description, payment type, budgets/hours* fields that map well to proposal elements (deliverables, pricing) and numeric estimates. Combine the **Asaniczka Upwork (50K)** and **Freelancer.com (9K)** datasets for broad coverage. (2) **Government/Contract Awards & Tenders** – e.g. World Bank/World Bank Group contracts and US procurement (City of Austin Plan-Holders) containing project scopes, vendors, and award amounts. These supply *structured budget and scope data* and serve as RAG documents for proposals. (3) **Software Effort Estimation Datasets** – classical COCOMO/NASA/desharnais datasets of code size vs. effort [2] [3]. They contain numeric *KLOC, FP, person-hours* that align with estimating time/budget. Augment with summarization corpora like **BillSum (US bills)** [4] (~23.5K bills with text+summary) to train the "weekly update" summarization task.

These cover our output schema: job postings yield deliverables and pricing (structured JSON); procurement/contract datasets yield budgets and timelines; estimation datasets yield numeric calibration; and summarization data improve narrative generation.

Recommended next steps: Download and inspect the above datasets (via Kaggle, HuggingFace, or data.gov links), extract relevant columns, anonymize as needed, and start converting rows into `{input, target}` JSONL pairs via pandas.

Below is a ranked list of candidate datasets, with details for each.

- **Upwork Job Postings (Asaniczka 2024, 50K records)** – *Job title, description, date, payment type, budget/hourly, country* [1]. CSV format (~24 MB); license ODC-By (open data). Example schema: `title`, `description`, `posted_date`, `budget`, `currency`, `payment_type`. Sample row: *"Title: Build mobile app; Description: We need an Android/iOS app for e-commerce; Budget: 5000; Currency: USD; Payment type: fixed"*. This maps well to our JSON: the `title` ↦ proposal title, `description` ↦ deliverables list and project summary (needs manual parsing or annotation to extract deliverables/timeline), `budget` and `payment_type` ↦ pricing object. Preprocessing: remove PII, normalize currencies/units (e.g. "USD 5000"→{"amount":5000,"currency":"USD"}). Annotation: likely need to tag deliverables and milestones from free-text (medium effort). *Annotation effort:* moderate (we'd parse descriptions into structured fields). Use-case: (A) pretrain on public data. No large documents (RAG: no). Split ~80% train / 10% val / 10% test (≈40K/5K/5K). **Example conversions:**
- *Input:* "Job Title: Web Developer. Description: "Build a WordPress site with ecommerce functionality, 5 pages including blog; must integrate payment gateway; deliver fully tested site."
  Output (JSON target):*

```
{
  "title": "Web Developer",
  "deliverables": ["WordPress e-commerce website", "5 pages including
```

```
blog", "Payment gateway integration"],
  "timeline": {"milestone1": "Design (2 wks)", "milestone2": "Development
(4 wks)","milestone3":"Testing & launch (1 wk)"},
  "pricing": {"currency":"USD", "amount":3000, "breakdown": {"design":
500,"dev":2000,"testing":500}},
  "payment_terms": "50% upfront, 50% on delivery",
  "summary": "Develop WordPress site with e-commerce, delivered in 7
weeks."
}
```

- *Input:* "Title: iOS App Developer. Desc: "Create an iPhone app for booking appointments. Includes UI design, backend API integration, database." Budget: 8000 USD (fixed)."
  Output: JSON with `title":"iOS App Developer"`, deliverables = ["iPhone booking app","UI design","API integration","Database setup"], estimatedHours/budget, milestones, etc.
- *Input:* "Title: Logo Design. Desc: "Design logo for new startup; two concepts and final deliverables in PNG and SVG." Payment: hourly, $50/hr."
  Output: JSON with deliverables = ["Logo concepts", "Final logo (PNG, SVG)"], pricing = {"currency":"USD","hourly_rate":50}, estimatedHours, terms etc.

Numeric caveats: Upwork budgets sometimes in ranges or hourly. Normalize "$30-$50/hr" to min/max. Watch currency conversion. **Biases:** Tech-focused (lots of dev/design jobs), mostly Western markets; mitigate by up/down-sampling sectors. **Snippet:**

```
df = pd.read_csv('upwork_jobs.csv')
df = df.dropna(subset=['title','description'])
output = []
for _, row in df.iterrows():
    input_text = f"Job Title: {row['title']}. Description: {row['description']}"
    target = {"title": row['title'], "deliverables": [], "timeline": [],
"pricing":{}, "payment_terms": "", "summary": ""}
    output.append({"input": input_text, "target": json.dumps(target)})
pd.DataFrame(output).to_json('upwork_in2out.jsonl', orient='records',
lines=True)
```

Combination: Combine with other freelancing datasets (see below) to diversify skills and budgets. Likely upsample niche categories if needed.

- **Freelancer.com Jobs (IsaacOresanya Kaggle, 9,193 records)** – *Scraped Freelancer.com postings in "Data Analysis" category*. CSV (~27 MB); license MIT [5] (commercial use allowed). Fields: `Job Title`, `Job Description`, `Payment Type`, `Minimum Budget`, `Maximum Budget`, `Skills`, etc. Sample: *"Title: Virtual Assistant; Description: Seeking VA for data entry and web search tasks..."* with budgets. Matches our schema similarly (title, desc → deliverables and summary; budget range → pricing breakdown). Size ~9K rows, format CSV. Many records may have ranges (e.g. "$200-$300"). Preprocess budgets and skill tags. Requires annotation for deliverables/timeline. *Annotation effort:* moderate. Use-case: (A) pretrain (small but useful variety). RAG: no. Split ~80/10/10. **Examples:**

- *Input:* “Job Title: Virtual Assistant. Desc: “We need a VA for data entry, email management, and research. Part-time project (20 hours/week).” Budget: 300 USD.”
  Output: JSON with deliverables=["Data entry","Email management","Research"], timeline (~2-3 weeks at 20 hr/wk), budget, etc.
- *Input:* “Title: Python Developer. Desc: “Develop script to clean and analyze data. Deliver cleaned datasets and report.” Fixed Budget: $1000.”
  Output: JSON with deliverables=["Python data-cleaning script","Clean datasets","Analysis report"], timeline, pricing.
- *Input:* “Title: Translator. Desc: “Translate 5-page document from English to Spanish.” Hourly, $25/hr.”
  Output: JSON with estimatedHours, deliverable, pricing.

Numeric caveats: Min/Max budgets present – can sum or pick range. **Biases:** “Data analysis” only; try merging categories. Some fields might have PII (client profiles); drop those. **Snippet:**

```
df = pd.read_csv('freelancer_data_analysis.csv')
df = df.dropna(subset=['Job Title','Job Description'])
df['input'] = "Title: " + df['Job Title'] + "; Desc: " + df['Job Description']
df['target'] = df.apply(lambda r: json.dumps({
    "title": r['Job Title'],
    "deliverables": [],
    "timeline": {},
    "pricing": {"amount_min": r.get('Minimum Budget'), "amount_max":
r.get('Maximum Budget'), "currency": "USD"},
    "payment_terms": r.get('Payment Type', ""),
    "summary": ""
}), axis=1)
df[['input','target']].to_json('freelancer_in2out.jsonl', orient='records',
lines=True)
```

Combine: Augment with Upwork data; downsample if too redundant.

- **All-Upwork Monthly Tracker (Asaniczka Kaggle, 240K+ records)** – *Hourly scrape of Upwork postings (Feb–Mar 2024)* [1]. CSV (67 MB); license ODC-By. Fields (from Gigasheet): `title, link, published_date, is_hourly, hourly_low, hourly_high, budget, country` [1]. No full descriptions, but titles plus payment fields. Useful for numeric/budget modeling (large size). Direct mapping: `title` → proposal title, `is_hourly/budget` → pricing, `country` (context). Needs enrichment (could fetch descriptions, but if not, treat title as short desc). *Size:* ~240K rows, suitable for (A) pretrain numeric/format learning. RAG: no. Split large (e.g. 192K/24K/24K). **Examples:**
- *Input:* “Title: WordPress Site Setup. Hourly: Yes. Rate: $20-$30/hour.”
  Output: JSON with estimatedHours (say 100), pricing, etc.
- *Input:* “Title: Research Data Analysis. Budget: $1500 fixed.”
  Output: JSON with pricing {"amount":1500,...}, deliverables inferred from title (“Data analysis report”), etc.
- *Input:* “Title: Mobile Game Developer. Hourly: Yes. Rate: $60-$80/hr.”
  Output: JSON with high estimatedHours, deliverables = ["Mobile game app"], etc.

Annotation: Low (no descriptions). May skip deliverables or infer from title. *Caveat:* Very terse; treat as weak supervision. Could upsample or downsample to balance hourly vs fixed. **Snippet:**

```
df = pd.read_csv('upwork_monthly.csv')
df['input'] = "Title: " + df['title'] + "; " + ("Hourly: " if df['is_hourly']
else "Budget: ")
df['input'] += df.apply(lambda r: f"${r['hourly_low']}-{r['hourly_high']}/hr" if
r['is_hourly'] else f"${r['budget']}", axis=1)
df['target'] = df.apply(lambda r: json.dumps({"title": r['title'],
"deliverables": [], "pricing": {}, "summary": ""}), axis=1)
df[['input','target']].to_json('upwork_monthly.jsonl', orient='records',
lines=True)
```

Combine: Merge with the 50K Upwork dataset; use the larger set to pretrain on numeric fields and text patterns.

- **Freelance Contracts (Asaniczka Kaggle, 1.3M entries)** – *Completed contract records from a freelancing platform; fields likely include: contract_id, buyer_id, seller_id, status, total_payment, currency, duration, start_date, end_date, project_title, category* [6]. CSV (67 MB); license ODC-By (as per snippet). This large set has *actual prices and durations* (often hourly vs fixed flagged). It provides real payment amounts and effort, useful to teach budget/time normalization. Example row: `{project_title: "Web design", total_payment: 800, currency: "USD", duration_hours: 40}`. Fields: likely no description, but has `project_title` and `category`. Use as (A) pretrain numeric. Map: `project_title` → partial summary, `total_payment` → estimatedBudget, `duration_hours` → estimatedHours. Minimal annotation: maybe none. RAG: no. Split (e.g. 1M train, 150K val, 150K test). **Examples:**
- *Input:* "Project Title: E-commerce Site. Total Payment: 1200 USD. Duration: 30h." *Output:* `{"estimatedBudget":1200,"currency":"USD","estimatedHours":30,"summary":"Develop e-commerce website"}`.
- *Input:* "Project: Logo Design; Payment: 300 EUR; Hours: 6." *Output:* JSON with numeric fields.
- *Input:* "Fixed-price contract 'SEO optimization', $500, 10h." *Output:* JSON with `"pricing":{"amount":500,"currency":"USD"}`, etc.

Caveats: Some projects are poorly described; currency conversion needed if multi-currency (normalize to USD). **Bias:** Likely dominated by IT categories; may underrepresent others. **Snippet:**

```
df = pd.read_csv('freelance_contracts.csv')
df = df[['project_title','total_payment','currency','duration_hours']]
df.columns = ['title','amount','currency','hours']
df.dropna(inplace=True)
df['input'] = "Project Title: " + df['title'] + ". Payment: " + df['currency'] +
" " + df['amount'].astype(str)
df['target'] = df.apply(lambda r: json.dumps({"estimatedBudget":
float(r['amount']), "currency": r['currency'], "estimatedHours":
```

```
float(r['hours'])}), axis=1)
df[['input','target']].to_json('contracts.jsonl', orient='records', lines=True)
```

Combine: Use alongside Upwork data to relate posted vs actual budgets. Possibly downsample if redundant large.

- **City of Austin "Plan Holders" (Data.gov API)** – *Local government bidding/proposal data*. CSV/JSON from data.austintexas.gov [7]. Fields: `vendor_name, proposal_request, project_number, let_type, date_received`, etc. "proposal_request" is often a textual RFP summary (deliverables needed). Sample: `{vendor_name:"ACME Co", proposal_request:"Upgrade city park lighting to LED", project_number:"P12345", let_type:"Construction"}`. License: US PD (government). Size: small (likely hundreds of rows). Map `proposal_request` → deliverables/scope, `let_type`/`project_number` → classification. Good (B) fine-tune on domain-specific proposals. Preprocessing: remove vendor names, anonymize. Annotation: minimal (just mapping fields). *Example Prompt:* "Request: Upgrade park lighting to LEDs, project number P12345; Vendor: ACME Co." Output JSON with `title:"LED park lighting upgrade", deliverables: [...], timeline (if any), pricing (if known or estimate), summary`. Example see *Examples below*. RAG: Yes (these are actual procurement texts). **Examples:**
- *Input:* "Proposal Request: 'Renovate downtown library; install new HVAC system and lighting.' Let Type: Construction."
  Output: JSON with deliverables=["New HVAC installation","Lighting replacement","Demolition and reconstruction work"], timeline estimated, etc.
- *Input:* "Project P678: 'Acquire 10 new patrol vehicles; include GPS units and decals.' Vendor: City Police Dept."
  Output: JSON with deliverables=["10 patrol vehicles","GPS units","Decals"], pricing=approx. (if known), summary.
- *Input:* "RFP: 'Develop IT ticketing system, integrate with existing ITSM.' Type: IT."
  Output: JSON structured accordingly.

Snippet: (using HTTP API via pandas)

```
import requests
res = requests.get('https://data.austintexas.gov/api/views/jd6h-b87p/rows.json?accessType=DOWNLOAD')
data = pd.DataFrame(res.json()['data'], columns=res.json()['meta']['view']['columns'])
data = data.rename(columns={"vendor's name":"vendor","proposal request":"request","State Project Number":"project_number"})
data['input'] = "Request: " + data['request']
data['target'] = data.apply(lambda r: json.dumps({
      "title": r['request'].split('.')[0],
      "deliverables": [], "timeline": {}, "pricing": {}, "summary": ""
}), axis=1)
```

```
data[['input','target']].to_json('austin_planholders.jsonl', orient='records',
lines=True)
```

Combine: Augment with other RFP/procurement texts. Likely downsample as small.

- **Software Effort Estimation (PROMISE/UCI/IBSG datasets)** – *Classic projects data* (COCOMO-81, NASA93, Desharnais, UCP, etc) [3] [2]. These are small CSVs (tens of rows each). Fields like LOC/KLOC, Function Points, etc → actual person-hours. *E.g.*, COCOMO-81 (63 rows, fields `KLOC`, `person_months` [2]), Desharnais (80 rows, `FP`, `Effort hours` [3]). License: public (research data). Useful for numeric calibration. Map input (e.g. "KLOC: 200, FP: 150") to JSON `estimatedHours` or `estimatedBudget`. Minimal annotation needed. Use-case: (A) pretrain numeric reasoning. RAG: no. Example prompt: "KLOC=120, use-case points=80" → output JSON `{"estimatedHours":1234,...}`. **Examples:**
- *Input:* "LOC: 30000, Complexity: Medium, TeamSize: 5" (from COCOMO-81)
  Output: `{"estimatedHours": 480, "rationale":"Based on COCOMO 1 model"}.`
- *Input:* "Function Points: 500, Adjusted UCP: 600" (from UCP dataset)
  Output: hours/budget accordingly.
- *Input:* "Estimated KLOC: 200, Actual Effort: 55 person-months" (Desharnais)
  Output: `{"estimatedHours": 55*160, "actualHours":55*160}`.

Caveats: Small size → overfit; use for numeric logic, not general language. Normalize scales (KLOC vs LOC).
Snippet:

```
df = pd.read_csv('Desharnais.csv')
df['input'] = "Function Points: " + df['FP'].astype(str) + "; Complexity: " +
df['DA'].astype(str)
df['target'] = df.apply(lambda r: json.dumps({"estimatedHours": r['Effort'],
"actualHours": r['Actual']}), axis=1)
df[['input','target']].to_json('desharnais.jsonl', orient='records', lines=True)
```

Combine: Merge multiple estimation tables into one large set; consider weighting to avoid small-sample noise.

- **BillSum (FiscalNote, 23.5K US Bills)** [4] – *US legislative bills (text + summary)*. Parquet/JSON; public domain (US federal bills are government works). ~23,455 rows (18.9K train + 3.3K test) [4]. Fields: `text` (long bill text), `summary` (Abstractive summary), `title`. Good for (A) training summarization style (weekly update summaries). Also RAG: the bill texts are long documents (up to ~5K tokens) that can be indexed and retrieved. Embedding: use sentence-transformers/all-MiniLM-L6-v2 (384d) on bill text. Example mapping: input = bill `text`, output = JSON `{"summary": [bill summary], "title": [bill title]}` or even break into our structure (`weekly_summary`, `accomplishments`, etc., if teased out). **Examples:**
- *Input:* Full text of a bill on cybersecurity funding.
  Output: `{"weekly_summary":"This bill allocates funding for cybersecurity initiatives...","title":"Cybersecurity Funding Act","deliverables":[],"metrics":{},"next_steps":""}`.

- *Input:* Bill to upgrade infrastructure at state parks.
  Output: JSON summarizing key provisions as summary/next steps.
- *Input:* Healthcare reform bill text.
  Output: JSON with `summary` capturing main aim, `deliverables` maybe blank, metrics empty.

Caveats: Abstract structure not matching our tasks exactly; may need prompts to rephrase into our JSON schema. But useful for narrative. **Snippet:**

```
import pandas as pd
import json
df = pd.read_parquet('billsum.parquet')
df['input'] = df['text']
df['target'] = df.apply(lambda r: json.dumps({
    "weekly_summary": r['summary'], "title": r['title'], "deliverables": [],
"metrics": {}, "next_steps": ""
}), axis=1)
df[['input','target']].to_json('billsum.jsonl', orient='records', lines=True)
```

Combine: Use with CNN/DailyMail or XSum (if license OK) for more summarization diversity; although focus on English technical tasks.

- **Google/Upwork/GitHub Jobs (Misc Kaggle)** – Additional job listing sets (e.g. *Monster.com Jobs* [8], *Dice.com Jobs*). These contain `job_description`, `title`, `salary`. Kaggle's PromptCloud jobs sets (~20k). License unclear (scraped content – likely CC BY?). Use if license is CC BY-SA (Kaggle's default). They add variety. Map like other job posts.

- **Task/Issue Tracking (GitHub Issues, Jira)** – E.g. HuggingFace "Github issues dataset" (issues from software repos). These have `title`, `body`, `labels`. Useful for converting issues to summaries or proposals. License: depends on source (public repos, so likely MIT/Apache). Use for training `weekly_summary` style outputs. RAG: indexing issues docs. Example: input = issue title/body, output JSON with summary of problem and proposed fix. Possibly (B) fine-tune on domain tasks.

- **Synthetic Data (Augmentation):** Generate "toy" project posts via templates. E.g. use GPT-4 to rewrite job postings as proposals or vice versa. E.g. "Convert this job ad into a structured proposal JSON" to create extra training examples. Or fill in proposals given skeleton (with random budgets/hours).

Suitability & Sizing:
- Upwork/ Freelancer jobs: *Pretrain (A)* – high priority for Phase A to learn language of proposals.
- Contracts/Procurement: *(A)* – good for numeric patterns and formal structure.
- Effort Estimation: *(A)* – small numeric data to calibrate.
- Summarization (BillSum): *(A)* for narrative skills and RAG knowledge.
- Combining: Yes – e.g. Jobs + Contracts for rich proposals; estimation + jobs for numeric balancing.

Embedding for RAG: Use `sentence-transformers/all-MiniLM-L6-v2` (384d) to embed large text docs (e.g. bill texts or RFPs) [4].

Model Architecture: For Phase A, fine-tune a *Flan-T5* model (e.g. Flan-T5-Large or XXL) on these sequences-to-JSON pairs. Flan-T5 is seq2seq and handles structured outputs. Also consider a decoder-only model (e.g. LLaMA2-7B) with LoRA adapters to learn to produce JSON. Flan-T5 (770M–3B params) can capture the structure well; LLaMA2-7B LoRA could later ingest longer context (for RAG).

Minimal/Ideal Data Volume: Aim for **>100K training pairs** as a minimal Phase-A corpus, ideally **500K–1M** for robust performance. The combined freelance/job datasets alone approach 300K+ rows, which is a good start.

Example Prompt/JSON Template (for Upwork job):
Input: "Job Title: *CRM Developer*. Description: *Build a CRM system with contacts, leads, and reports; integrate email; deliver user manual.* Payment: *Hourly $40/hr*."
Target:

```
{
  "title": "CRM System Development",
  "deliverables": ["CRM software with contacts/leads management","Email integration","User manual"],
  "timeline": {"design": "2 weeks","development": "4 weeks","testing": "1 week"},
  "pricing": {"currency":"USD","hourly_rate":40},
  "payment_terms": "Bi-weekly payments on milestone completion",
  "summary": "Develop and deliver a CRM application over 7 weeks."
}
```

Bias and Quality Notes: Many jobs are IT/tech-heavy; ensure to balance industry/domain if possible. Government data (bills, contracts) are US-centric. Remove personal identifiers (names, contacts) from job descriptions.

Next Steps:

1. **Dataset Download:** Use Kaggle API or browser to download the Upwork, Freelancer, and other relevant CSV/JSON files. Use the data.gov API (or download link) for Austin Plan Holders. Use HuggingFace's `datasets` library to fetch BillSum.
2. **Preprocessing:** In a Jupyter notebook, load each dataset into pandas. Drop or mask any PII. Normalize numeric fields (convert all currencies to e.g. USD, hours to floats).
3. **Annotation/Conversion:** Define mapping rules: e.g. `title` → `title`, parse `description` into `deliverables` list (maybe via heuristic splitting or manual rules), map `budget/hourly` → `pricing` JSON. Write pandas transformations as above to produce JSONL lines of `{"input": ..., "target": ...}`.
4. **Split:** Randomly split each dataset into train/val/test (e.g. 80/10/10). Possibly reserve a combined test set.
5. **Integration:** Concatenate all JSONL sources into a single Phase-A corpus. Verify format.
6. **Vector DB Prep (RAG):** For datasets like BillSum (long docs), create embeddings using `sentence-transformers/all-MiniLM-L6-v2` (384d) and index with e.g. FAISS for retrieval.

These steps will prepare the Phase-A training data for our seq2seq model.

Sources: We derived dataset details from Kaggle descriptions and documentation [1] [3] [2] [4], as well as data.gov metadata [7]. Each suggested dataset provides either direct fields or content that can be mapped/annotated to our target JSON schema.

---

[1] Comprehensive Tracker: Upwork Job Postings Updated Monthly (200k+) | Spreadsheet Download | Gigasheet
https://www.gigasheet.com/sample-data/upwork-jobs

[2] [3] GitHub - Derek-Jones/Software-estimation-datasets: Collected public software estimation datasets
https://github.com/Derek-Jones/Software-estimation-datasets

[4] FiscalNote/billsum · Datasets at Hugging Face
https://huggingface.co/datasets/FiscalNote/billsum

[5] Freelancer Data Analysis Jobs Dataset - Kaggle
https://www.kaggle.com/datasets/isaacoresanya/freelancer

[6] Freelance Contracts Dataset (1.3 Million Entries) - Kaggle
https://www.kaggle.com/datasets/asaniczka/freelance-contracts-dataset-1-3-million-entries

[7] Dataset - Catalog
https://catalog.data.gov/dataset/?tags=proposal

[8] US jobs on Monster.com - Kaggle
https://www.kaggle.com/datasets/PromptCloudHQ/us-jobs-on-monstercom