AP Bill Worker – Gemini Parameters & Parsing Logic Reference

Reference for Gemini parameters, extraction schemas, and all parsing logic used in this project. **Assumes Odoo 19.**

Gemini Configuration

Env Variable	Default	Description
`GEMINI_API_KEY`	*(required)*	Google AI API key
`GEMINI_MODEL`	`gemini-3-pro-preview`	Primary model for both passes
`GEMINI_FALLBACK_MODEL`	`gemini-2.5-pro`	Stable fallback if primary fails

Both passes use **structured JSON output** (`responseMimeType: "application/json"` + `responseSchema`).

Pass 1: Invoice Extraction

Function: `extractInvoiceWithGemini(ocrText, config, attachment)`

Input: OCR text + **image or PDF bytes** (for multi-modal accuracy). Both `image/*` and `application/pdf` mimetypes are sent to Gemini as inline data so the model can read the document visually. PDFs were previously OCR-only; now the raw PDF is sent for better extraction.

Output schema – all fields below are extracted in a single Gemini call:

vendor

Field	Type	Description
`name`	string	Best-guess vendor/seller name
`confidence`	number	0–1 confidence score
`source`	string	Where the name was found: `header`, `body`, `atp_printer_box`, `unknown`

vendor_candidates

Array of up to 5 alternative vendor guesses, each with `name`, `confidence`, `source`.

vendor_details

Field	Type	Description
`tin`	string	Tax Identification Number (PH format)
`branch_code`	string	Branch code if present

Field	Type	Description
`address`	string	Vendor address
`entity_type`	string	`corporation`, `sole_proprietor`, `individual`, `unknown`
`trade_name`	string	Business/DBA name (e.g. "JORJEL LAUNDRY SHOP")
`proprietor_name`	string	Owner's personal name for sole proprietors (e.g. "JOCELYN E. SANTOS")

Entity type detection rules:

- `corporation` – name ends with Inc., Corp., Co., LLC, etc.
- `sole_proprietor` – both a trade name AND a personal owner name present (look for "Prop.", "Owner:")
- `individual` – vendor is clearly a person with no business name
- `unknown` – cannot determine

invoice

Field	Type	Description
`number`	string	Invoice/receipt number
`date`	string	Date in `YYYY-MM-DD` format
`date_confidence`	number	0–1
`currency`	string	ISO currency code (e.g. "PHP", "USD")

vat

Field	Type	Description
`classification`	string	`vatable`, `exempt`, `zero_rated`, `unknown`
`goods_or_services`	string	`goods`, `services`, `unknown`
`vatable_base`	number	Net amount before VAT
`vat_amount`	number	VAT amount
`exempt_amount`	number	VAT-exempt amount
`zero_rated_amount`	number	Zero-rated amount
`evidence`	string	Text snippet supporting the classification

VAT classification rules:

- `exempt` – receipt shows "VAT Exempt" or VAT-exempt amount
- `zero_rated` – receipt shows "Zero Rated", "0% ZR"
- `vatable` – shows VAT amount, "VAT Sales", "Vatable Sales", or 12% VAT
- `unknown` – none of the above

totals

Field	Type	Description
`grand_total`	number	Final amount due (VAT-inclusive if applicable)
`net_total`	number	Amount BEFORE VAT. Computed as `grand_total / 1.12` if only inclusive total shown
`tax_total`	number	Tax amount
`amounts_are_vat_inclusive`	boolean	`true` if prices already include 12% VAT
`vat_exempt_amount`	number	Exempt portion
`zero_rated_amount`	number	Zero-rated portion

All totals have accompanying `*_confidence` fields (0–1).

line_items

Array of extracted line items:

Field	Type	Description
`description`	string	Item description
`quantity`	number	Qty
`unit_price`	number	Price per unit
`amount`	number	Line total
`unit_price_includes_vat`	boolean	Whether unit price is VAT-inclusive (overridden by global `amounts_are_vat_inclusive` when true)
`expense_category`	string	One of: `office_supplies`, `meals`, `repairs`, `rent`, `fuel`, `professional_fees`, `freight`, `utilities`, `inventory`, `other`

expense_account_hint

Field	Type	Description
`category`	string	Best-fit category (same options as line_items)
`suggested_account_name`	string	Human-friendly account name guess
`confidence`	number	0–1
`evidence`	string	Supporting snippet

amount_candidates

Array of all notable amounts found, each with `label`, `amount`, `confidence`, `snippet`. Used by `fixExtractedAmounts`.

warnings

Array of strings flagging potential issues (e.g. "Multiple totals found", "Low confidence vendor match").

Amount Correction (`fixExtractedAmounts`)

Post-processing corrects Gemini misreads for handwritten/low-quality receipts:

Case	When	Action
A	Line sum >> grand total (e.g. 10500 vs 1045)	Use line sum
B	Grand total >> line sum (e.g. 85509 vs 8017)	Use line sum or best amount_candidate
C	Amount candidate >> grand total (truncated)	Use candidate
D	Grand total >> amount candidate (inflated)	Use candidate
E	OCR max >> grand total (truncated)	Use OCR max
F	Grand total >> OCR max (inflated)	Use OCR max
G	Decimal misread (grandTotal/10 or /100 ≈ line sum)	Use line sum or candidate
H	grand_total ≈ tax_total or vat_amount (VAT picked as total)	Use vatable_base + tax, or best "total" candidate, or derive from 12%
I	tax_total > 20% of grand_total (impossible for PH 12% VAT)	Use best total candidate, line sum, or derive from tax / 0.12 × 1.12

PH-Specific Prompt Rules

- **ATP/Printer box exclusion**: Names found near "ATP", "BIR Permit", "Printer", "Accreditation" are excluded as vendor candidates.
- **VAT-inclusive detection**: Most PH receipts show prices including 12% VAT. The prompt instructs Gemini to detect this and compute `net_total = grand_total / 1.12`.
- **Grand total vs line items**: When totals disagree, prefer the reading where arithmetic is consistent. Never pick a grand total 5x–15x the line item sum (likely misread).
- **VAT vs total (critical)**: grand_total must NEVER be a VAT/tax component. "VAT Amount: 428.57" means the tax is 428.57, NOT the total. If grand_total ≈ tax_total, the wrong number was picked. If tax_total > grand_total, the grand_total is wrong (tax cannot exceed total).
- **Extract ALL line items**: Do not skip items. If the invoice lists 10 products, extract all 10. Re-examine if only one line item was found but the document clearly has more.
- **Sole proprietor detection**: Gemini looks for "Prop.", "Owner", or personal names near trade names.

Pass 2: Account Assignment

Function: `assignAccountsWithGemini(extracted, expenseAccounts, config, industry, ocrText)`

Input:

- Extracted data from Pass 1
- Full chart of expense accounts from Odoo (`account.account` where `account_type in [expense, expense_direct_cost, expense_depreciation, asset_current]`)
- Company industry (see Industry Resolution)
- OCR text (for additional context)

Output schema:

Field	Type	Description
`assignments`	array	Per-line account picks
`assignments[].line_index`	number	0-based index into line_items
`assignments[].account_id`	number	Matched Odoo account ID
`assignments[].account_code`	string	Account code
`assignments[].account_name`	string	Account name
`assignments[].confidence`	number	0–1
`assignments[].reasoning`	string	Why this account was chosen
`assignments[].alternatives`	array	Fallback account picks
`bill_level_account_id`	number	Best single account for the whole bill
`bill_level_confidence`	number	0–1

Industry in Account Selection

When `industry` is non-empty, the prompt includes:

```
COMPANY INDUSTRY: {industry}
Use industry context to decide COST OF REVENUE vs OPERATING EXPENSE:
- Direct use in core product/service → Cost of Revenue / COGS / Cost of Sales
- Back-office, admin, support → Operating Expense
- Examples: Restaurant (ingredients→COGS, cleaning→OpEx), Retail
(merchandise→COGS), Laundry (detergent→COGS), etc.
```

Prompt Rules

1. **Specificity over generality**: NEVER pick "Admin Expense", "Miscellaneous", "General Expense" unless no specific account exists.
2. **Match by item description, not vendor**: What was bought matters, not who sold it.
3. **Industry-aware COGS vs OpEx**: When industry is known, use it to decide Cost of Revenue vs Operating Expense.
4. **Philippine accountant persona**: Think like a PH accountant recording a vendor bill.

Item-to-Account Examples (in prompt)

Item	Account
TABLE CLOTH	Supplies / Housekeeping Supplies
LPG REFILL 11KG	Fuel & Oil / Gas & Oil
BOND PAPER A4	Office Supplies / Stationery
TONER CARTRIDGE	Office Supplies / Printing
ELECTRICITY BILL	Utilities / Power & Light
JANITORIAL SUPPLIES	Janitorial / Cleaning Supplies
FOOD / MEALS	Meals & Entertainment
LEGAL FEES	Professional Fees
SHIPPING / DELIVERY	Freight / Shipping
FABRIC / CLOTH	Raw Materials (or COGS for mfg)
DETERGENT / BLEACH	Janitorial (or COGS for laundry)

Industry Resolution

Sources (checked in order):

1. **SOURCE Odoo – General task**
   - Project IDs from routing sheet `source_project_id`
   - Task: name = "General", stage = "General"
   - Field: `x_studio_industry` (or `SOURCE_GENERAL_TASK_INDUSTRY_FIELD`)
   - Requires: `SOURCE_BASE_URL`, `SOURCE_DB`, `SOURCE_LOGIN`, `SOURCE_PASSWORD` (Secret Manager)
2. **TARGET Odoo**
   - `res.company.x_studio_industry`
   - `res.company.industry_id`
   - `res.partner.industry_id` (via company's partner)

Industry is written to the routing sheet `industry` column and passed to `assignAccountsWithGemini` for account selection.

Expense Account Loading (Odoo 19)

Function: `loadExpenseAccounts(odoo, companyId)`

Query strategy (cascading, no `company_id` filter on `account.account`):

1. `account_type in [expense, expense_direct_cost, expense_depreciation, asset_current]`
2. `account_type in [expense, expense_direct_cost]`

3. `code like "5" OR code like "6"`
4. All accounts

Each attempt is logged. Company scoping via `kwWithCompany(companyId)` context.

Vendor Resolution

Search Order (`findVendor`)

1. Search Odoo `res.partner` by **primary vendor name** (from `vendor.name`)
2. Search by **trade name** (from `vendor_details.trade_name`)
3. Search by **proprietor name** (from `vendor_details.proprietor_name`)

All searches: `name ilike <value>` + `supplier_rank > 0`.

Auto-Creation (`createVendorIfMissing`)

Conditions:

- Vendor confidence >= 0.9
- Not an ATP/printer vendor
- No existing match found

Sole proprietor handling:

- If `entity_type` is `sole_proprietor` or `individual` AND `proprietor_name` is present → Odoo vendor created with **proprietor's personal name**
- Trade name goes into `comment` field
- Sets `company_type: "person"` (falls back to `is_company: false`)

Corporation handling:

- Vendor created with business name
- Sets `company_type: "company"`

Fields written:

- `name`, `supplier_rank: 1`, `street`, `vat`, `comment`

Tax & VAT Logic

Tax ID Selection (`pickTaxIds`)

Based on `vat.classification` from Gemini:

- `exempt`, `zero_rated`, `unknown` → **no tax IDs**
- `vatable` → picks tax IDs from routing config based on `goods_or_services`

Tax ID sources (from routing sheet, auto-resolved from Odoo):

- `vat_purchase_tax_id_goods` – for goods purchases

- `vat_purchase_tax_id_services` – for service purchases
- `vat_purchase_tax_id_generic` – fallback

Tax Auto-Resolution (`pickVatTaxesForCompany`)

Queries `account.tax` where `type_tax_use = purchase`, 12% VAT.

Filters out: Capital goods taxes, withholding taxes, import taxes.

Prefers: `price_include = false` (adds VAT on top).

Tax Metadata (`getTaxMeta`)

Reads from Odoo: `price_include`, `amount` (rate %).

Price Adjustment (VAT-Inclusive)

Function: `adjustPriceForTax(price, invoiceVatInclusive, taxPriceInclude, taxRate)`

Invoice Price	Odoo Tax Config	Action
VAT-inclusive	`price_include = true`	No adjustment
VAT-inclusive	`price_include = false`	Divide by `(1 + rate/100)` to get net
VAT-exclusive	`price_include = true`	Multiply by `(1 + rate/100)`
VAT-exclusive	`price_include = false`	No adjustment

Multi-line VAT logic: When `amounts_are_vat_inclusive` is true, **all** line unit prices are treated as VAT-inclusive (`globalVatInclusive` overrides per-line `unit_price_includes_vat`). This avoids double VAT when Gemini mistakenly sets per-line to false.

Total Reconciliation (Grand Total Prevails)

Function: `buildBillVals` – ensures Odoo bill total matches extracted grand total.

1. **Expected untaxed**: `net_total` from extraction, or `grand_total / 1.12` if not provided.
2. **Multi-line path**: After building line items, sum their untaxed amounts. If diff from `expectedUntaxed` > 0.005 and < 15%, adjust the **last line's** `price_unit` to close the gap.
3. **Single-line path**: Use `expectedUntaxed` directly as `price_unit`.

Result: Odoo total = expected untaxed × 1.12 ≈ grand total.

Expense Account Cascade

Function: `resolveExpenseAccountId(...)` – 8-tier resolution:

Tier 1: Vendor Default

Read `property_account_expense_id` from matched `res.partner`. **Skip if that account is generic** (e.g. Admin Expense).

Tier 2: Gemini Pass 2

Use `account_id` from `assignAccountsWithGemini`. Validated and anti-generic (prefer non-generic alternatives).

Tier 3: Vendor Name Keywords

Match vendor name (e.g. "FABRIC TRADING") against account names.

Tier 4: Sheet Mapping (AccountMapping tab)

Lookup by `category` + `company_id` + `target_db`.

Tier 5: Fuzzy Name Match

Match line description + category keywords against account names. Penalizes generic accounts.

Tier 6: Gemini Last Resort

Use Gemini's primary pick even if generic (better than Odoo default).

Tier 7: Keyword Last Resort

Best non-generic account matching description/category/vendor keywords; else first non-generic; else first available.

Tier 8: Env Fallback

`DEFAULT_EXPENSE_ACCOUNT_ID` from environment.

Tier 9: None

Returns `accountId: 0` – Odoo uses its own default.

Bill Construction

Function: `buildBillVals(...)`

Line Items vs Single Line

- If extracted `line_items` exist AND their total is within 5% of invoice total → **itemized lines**
- Otherwise → **single summary line**

Per-Line Fields

Field	Source

Field	Source
`name`	**Itemized**: line item description (max 256 chars). **Single-line fallback**: extracted line item descriptions joined (if any); else `"Vendor Bill"`. Never uses `expense_account_hint.suggested_account_name` as the line label.
`quantity`	From extraction
`price_unit`	Adjusted via `adjustPriceForTax()`; last line may be tweaked for total reconciliation
`account_id`	From expense account cascade
`tax_ids`	`[[6, 0, taxIds]]`

Document Linking

Function: `linkDocumentToBill(odoo, companyId, docId, billId, logger)`

1. Reads original `folder_id` from the document
2. Sets `res_model = "account.move"`, `res_id = billId`, `account_move_id`, `invoice_id`
3. **Includes `folder_id` in the same write** to try to prevent Odoo from moving the document
4. **Retry logic** (800, 1500, 3000, 5000 ms): if folder was moved, restore it
5. Posts a clickable link to the document in the bill's chatter

Chatter Messages

Posted with `body_is_html: true` (Odoo 19). Messages include:

- Source document link
- Vendor extraction details
- Account suggestions per line (code, name, resolution tier)
- Extracted amounts (grand total, net total, tax, VAT-inclusive flag)
- Warnings (if vendor confidence < 0.9 or extraction warnings)

Reprocessing (Deleted Bills)

Marker Format

```
BILL_OCR_PROCESSED|V1|<target_key>|doc:<doc_id>|bill:<bill_id>|...
```

Reprocess Flow

1. Read marker from `ir.attachment.description`
2. If bill was **deleted**: clear marker, reprocess from scratch (`skip_duplicate_check` bypasses duplicate check)
3. If bill exists: skip

`run-one` Stale Link Cleanup

When document's `res_model` points to a deleted `account.move`: clears `res_model`, `res_id`, `account_move_id`, `invoice_id`, then reprocesses.

Retry & Fallback

Function: `geminiWithRetryAndFallback(config, body, options)`

1. Try **primary model** (`GEMINI_MODEL`)
2. On HTTP 429/500/503: retry up to 2 times with backoff (3s, 6s)
3. If primary exhausted: try **fallback model** (`GEMINI_FALLBACK_MODEL`)
4. Pass 1: throws on total failure
5. Pass 2: returns `null` on failure (cascade continues without Gemini)

Auto-Resolved Routing Fields

These columns in the ProjectRouting sheet are **auto-populated** on each run:

Column	Source
`vat_purchase_tax_id_goods`	12% VAT for goods (`account.tax`)
`vat_purchase_tax_id_services`	12% VAT for services
`vat_purchase_tax_id_generic`	Generic 12% VAT
`purchase_journal_id`	"Vendor Bills" journal (`account.journal`)
`ap_folder_id`	AP folder in Documents
`industry`	General task (`x_studio_industry`) or TARGET Odoo (`res.company`, `res.partner`)

Refreshed on each `getRoutingRows` and written back to the Google Sheet.