ChatGPT

DBOT System Technical Design Document

1. System Overview

Problem: The DBOT system aims to automate long-term **fundamental valuation** of any public company by mimicking the approach of renowned valuation expert Aswath Damodaran [1] [2]. Traditional fundamental analysis is time-consuming and human-intensive, while quantitative methods often ignore nuanced company specifics [3] [4]. DBOT addresses this gap by combining depth (detailed company-specific analysis) with breadth (scaling to thousands of companies) in an AI-driven framework [5].

Scope: This design focuses on implementing the multi-agent valuation pipeline described in *"DBOT: Artificial Intelligence for Systematic Long-Term Investing."* The system will ingest **public financial data**, perform **DCF (Discounted Cash Flow)** valuations using Damodaran's methodology, incorporate **analyst consensus**, **peer comparables**, and **latest news**, and produce a **comprehensive valuation report** for any publicly traded company. The initial implementation will be a Python-based prototype running on local or cloud servers, using **open/free data sources** (e.g. Yahoo Finance, Alpha Vantage, SEC EDGAR, Google News) and **hosted LLM APIs** (e.g. OpenAI GPT-4, Anthropic Claude) for language and reasoning tasks. The design ensures the system is **extensible to cloud-native deployment** (with modular agents, caching, and API-based data access) and can be scaled to cover large universes of stocks over time [6] [7].

Success Criteria: The system should produce **high-quality, stable valuation reports** that are comparable to a human expert's analysis. "High-quality" means the reports are factually accurate, logically reasoned, and include relevant data, charts, and source citations to support conclusions [8] [9]. "Stable" means the valuation outcome and recommendations are **consistent across runs** and not overly sensitive to minor prompt or data variations [10]. This stability allows results to be back-tested and trusted, akin to traditional quantitative strategies [10]. The output should explicitly state the **intrinsic value (v)** per share vs. the market price, key assumptions (growth, margins, etc.), and a clear recommendation (e.g. buy/sell or undervalued/overvalued). Success is measured by (a) the **accuracy and consistency** of valuations (e.g. DBOT's BYD valuation remained in the $417–$468 range through iterations [11]), (b) the **expert-like insight** in the narrative (e.g. highlighting industry context, competitive positioning [12] [13]), and (c) the inclusion of **complete justifications** (sensitivity analysis, peer comparisons, news impacts) similar to a professional analyst's report [14] [13].

Non-Goals: The system is **not** intended for high-frequency trading or short-term price prediction – it won't make intraday decisions or react to every minor market tick. It focuses on **long-term, fundamental value** and will ignore fleeting market sentiment that isn't grounded in fundamentals [15] [16]. It will not provide investment advice beyond the scope of valuation (e.g. portfolio allocation or risk management strategies are out of scope). Also, DBOT is not meant to be a fully autonomous trading agent executing trades; rather, it generates analysis to inform human decisions or systematic strategies. Finally, DBOT will **not violate data access rules** – it only uses publicly available data and will avoid insider information or any unethical data acquisition.

2. Architecture

Overview: DBOT is structured as a **multi-agent system** coordinated by a central **Supervisor (or Orchestrator) agent** [17] [18]. Each agent is a specialized AI component responsible for a distinct aspect of the valuation. Figure 2 of the source paper illustrates the architecture, where the Supervisor calls a sequence of agent sub-processes to incrementally build the valuation analysis [17] [19]. The agents communicate by passing data (e.g. financial inputs, intermediate valuations) and **"messages"** in natural language prompts that instruct the next agent what to do [20] [21]. This design leverages LLMs' ability to interpret complex instructions in English, effectively allowing us to "program" agent behaviors via prompts [21].

Component Diagram: *The DBOT system consists of the following main components (agents) and data flows:*

- **Supervisor Agent (Orchestrator):** Initiates the process, aggregates data, and manages control flow. It first triggers a **waterfall sequence** of valuation sub-agents (valuation, sensitivity, consensus, comparables, news), then enters an **iterative loop** where it asks an LLM (via a "router" prompt) which agent to call next based on context [22] [23]. Once convergence criteria are met, it calls the report-writing agents to finalize output [24] [25]. The Supervisor holds the central state: the current valuation inputs (I) and latest valuation result (v), updating them as agents return new information.

- **Valuation Agent (Quantitative Valuation):** Accesses the **financial model** (Damodaran's "Ginzu" DCF spreadsheet logic) to compute the company's intrinsic value [26]. It uses four key company-specific **value drivers** – *sales growth, operating margin, cost of capital,* and *reinvestment efficiency* – along with relevant macro factors (e.g. interest rates) to perform the valuation [26] [27]. The Valuation agent interacts with the **free cash flow valuation function (f<sub>fcf</sub>)**, a Python module that applies the DCF math. The agent can update input assumptions (within I) via an LLM prompt if needed (for example, to incorporate a scenario or new info) and then call f<sub>fcf</sub> to get a new v. **Inputs:** the current spreadsheet inputs I (with financial data and assumptions); **Outputs:** an updated I (if assumptions changed) and a computed valuation v [28] [29]. It uses no external APIs except the internal financial model and may use an LLM to assist in interpreting the model or ensuring consistency with Damodaran's methodology (the agent is initialized with Damodaran's own explanation of the "Ginzu" model for guidance [30]). *Caching:* The financial model results (v for a given set of I) can be cached per company to avoid recomputation if the same inputs are revisited. *Errors:* If the DCF function fails (e.g. due to division by zero or missing data), the agent returns an error message for the Supervisor to handle (possibly by using default assumptions or notifying the Critic). *Timeouts:* Minimal, as computation is local and fast (e.g. 5 seconds limit).

- **Sensitivity Analysis Agent:** Explores **alternative scenarios** by varying the value drivers and observing the impact on valuation [31] [32]. Uniquely, this agent uses the LLM to **"reason" over the spreadsheet**, meaning it can iteratively adjust inputs and check outputs to achieve a goal (for example, *"What growth rate would yield the current market price as fair value?"*). It might employ a strategy akin to a binary search or trial-and-error via prompt: e.g. *"Decrease the growth rate by 1% and see if v aligns closer to market"*. **Inputs:** current I and v, plus any target context (like market price); **Outputs:** a set of scenario outcomes, often in tabular form (e.g. a table of valuations under different growth/margin assumptions) [33]. The Valuation agent's calculation function (f<sub>fcf</sub>) is used repeatedly here; the Sensitivity agent essentially orchestrates multiple calls to f<sub>fcf</sub> with modified I values. *Prompt:* P<sub>sensitivity</sub> instructs the LLM to adjust one or more drivers and request the new valuation [34]. *Caching:* It may cache scenario results to avoid repeating identical adjustments. *Errors:* If an adjustment leads to nonsensical inputs (negative sales, etc.), the agent discards that scenario and tries a different approach, possibly guided by constraints in the prompt. *Timeouts:* Limit number of iterations (e.g. 5 trials) to avoid infinite loops in case the target cannot be reached. This agent's output (scenario table) is later fed to the Plotting and Writer agents for inclusion in the report.

- **Consensus Agent:** Gathers **analyst consensus estimates** and opinion data for the company [35]. This typically includes forecasts such as next-year earnings or revenue growth, target price, and possibly analyst rating distributions. **Inputs:** company identifier (Ct or Cn) and current I (to know what metrics to match); **Outputs:** updated I with consensus-derived inputs or a summary of consensus vs. DBOT's assumptions. For example, if analysts expect 5% growth but DBOT is assuming 7%, the agent might adjust or at least flag this discrepancy. It calls **external financial data APIs** – e.g. Yahoo Finance or Alpha Vantage – to retrieve consensus figures (Yahoo's API or site often provides analyst target price and growth estimates). All sources used are free; e.g. Yahoo's public data via `yfinance` (a free Python library that can fetch financial info easily [36]) or **Alpha Vantage** (which offers free JSON APIs for fundamental data [37]). *Prompt:* P<sub>consensus</sub> guides the LLM to incorporate analyst expectations into the valuation: *"Adjust the forecast inputs in I based on average analyst predictions and recalc value."* The agent might reduce the growth driver if analysts are pessimistic, etc. *Caching:* Daily or weekly caching of consensus data is used, since analyst estimates don't change intra-day. *Errors:* If no consensus data is available (e.g. for a very small company), the agent returns gracefully with no changes. *Timeouts:* 10–15 seconds for data fetch APIs (to handle network latency).

- **Comparables Agent:** Identifies **peer companies** and conducts a relative valuation check [35]. It finds companies in the same industry or with similar business models – often using industry codes or known competitors (Yahoo Finance's API can list peer tickers, or one can use a static mapping of sectors). **Inputs:** company ticker (Ct) and possibly industry classification; **Outputs:** a set of comparables with key multiples (e.g. EV/EBITDA, P/E) and an assessment of where the target company stands. For instance, DBOT's BYD analysis included a chart of BYD's Terminal EV/EBITDA (8.8x) against Tesla, Nio, etc., showing BYD was cheaper than peers [14]. The Comparables agent may output an updated I (if it decides to tweak assumptions after seeing peers – though typically comparables are more for context) and a text summary of relative valuation. It uses **open data** like Yahoo Finance or Alpha Vantage to fetch peer metrics, or SEC EDGAR if needed for specific financials. *Bias handling:* The agent takes care not to let outlier comparables overly sway the valuation – it looks at a reasonable peer group and might ignore extremes. *Prompt:* P<sub>comparables</sub>: *"Identify 3-5 comparable companies and summarize how our valuation multiples compare. If our stock's value is out-of-line, consider if adjustments to drivers are needed." Caching:* The list of peers for each company can be cached, as well as fetched ratios, updated perhaps quarterly. *Errors:* If peers cannot be found (e.g. a very unique company), the agent reports "no close comparables" and moves on. *Timeouts:* Similar to consensus agent for data fetch.

- **News Agent:** Scans recent **news and media** for any material information about the company, industry, or macro environment that could impact valuation [38] [39]. It performs a **task-focused summarization**, filtering only news relevant to the company's fundamentals (ignoring generic or irrelevant pieces) [40]. The news reading is done in **three phases** to mimic human analysis [41]: (1) read **headlines** only and note any important ones; (2) read headline + first paragraph of each article for more context; (3) fully read the most relevant articles. At each phase the agent “reflects” on whether this news might change any assumptions or value drivers [41]. **Inputs:** ticker or company name (for search query) and possibly the current I (to know what aspects might be sensitive – e.g. if I assumes high growth, look for news on sales or market expansion). **Outputs:** a concise **news summary** highlighting key developments and their potential effect on valuation (e.g. “New EV subsidy in Europe could boost BYD’s sales” -> might justify higher growth). This summary (text) is appended to I or kept separately to feed the Writer agent. The News agent interfaces with **free news sources/APIs**: for example, Google News RSS or Bing News search for the ticker (ensuring to use only publicly available content). It may access specific outlets (WSJ, FT, etc.) via web scraping or RSS feeds for headlines [42] [43]. *De-duplication:* It cross-checks headlines to avoid repeating the same story from multiple sources. *Factuality:* The agent uses the Critic (or an internal check) to verify that any numeric facts from news (e.g. “Q3 revenue grew 50%”) match the data in I or DC when possible, reducing hallucinations. *Prompt:* P<sub>news</sub>: *“Summarize recent news for [Company] that affects its financial outlook (growth, margins, risk). Focus only on fundamental impacts.”* After gathering raw news text, another prompt P<sub>news summary</sub> condenses it [44] [45]. The agent might also retrieve images from news (like a stock price chart or a relevant figure) to include if they enhance the narrative [46], though in this implementation we restrict to textual output for simplicity. *Caching:* News results can be cached per day to avoid repetitive API calls if the system is run multiple times. *Errors:* If news API fails or returns nothing, the agent will output “No significant recent news found.” *Timeout:* Perhaps 15–30 seconds (news scraping can be slow), with a limit on number of articles (e.g. read top 5 articles).

- **Plotting Agent:** Generates **visuals (charts/plots)** to support the valuation narrative [47] [48]. For example, it may create a chart of valuation sensitivity (e.g. a tornado chart or scenario bar chart), a comparative multiples bar chart (as in BYD’s TEV/EBITDA comparison [49] [14]), and possibly historical financial trends. This agent takes structured data (from I or DC) and produces image files. **Inputs:** I (especially any tables meant for presentation, like the output of sensitivity analysis or a list of comparables with their multiples). **Outputs:** paths or references to generated image files (to be embedded in the final report). Implementation uses Python plotting libraries (e.g. Matplotlib or Plotly) for reliability. An interesting approach (noted in the paper) is to use an LLM to generate Python code for the plot, then execute it to obtain an image [50] [51]. We can use this approach for flexibility: the agent’s prompt P<sub>plot</sub> might say: *“Write Python code to plot Company vs Peers EV/EBITDA as a bar chart using the given data.”* Then the code is run to produce the plot [50]. Alternatively, for MVP we might bypass the LLM and directly code standard charts for known use cases (e.g. always output a sensitivity table, so use a templated matplotlib plot). *Caching:* Not usually needed except to store the generated images. *Errors:* If code generation fails or produces an error, the agent can fall back to a template plot or output an error for the Critic to note. *Timeout:* ~10 seconds per plot generation.

- **Report Writer Agent:** Composes the **final valuation report** as a well-structured document [17] [52]. This agent has **read-only access** to all inputs and analysis results – it cannot change any values, only describe them [30]. It uses an LLM to weave together the narrative (“story”) and the “numbers” from the valuation model [30]. The Writer agent’s prompt P<sub>report</sub> instructs it to produce a comprehensive report given the assembled data (I, v, news summary, comparables, charts) [53] [54]. The output should include: an introduction (possibly with a catchy title summarizing the thesis, e.g. *“BYD in 2024: Riding the EV Wave or Struggling Through the Competitive Storm?”* [55]), a section for each major analysis (initial valuation, market comparison, sensitivity results, consensus discussion, news impacts), and a conclusion with the final valuation vs market price and recommendation. The Writer explicitly inserts placeholders or references for charts/tables (which the Plotting agent produced) and citations for data sources (e.g. "according to last annual report [12]"). **Inputs:** the final I and v, plus textual summaries from each agent (often the Supervisor will collate these into a single context for the writer LLM). **Outputs:** a formatted Markdown or HTML report R. The Writer uses a **factual style** guided by Damodaran's own writings (the system provides Damodaran's methodology video transcript or notes as part of prompt context to imbue the style [30]). *Length control:* The Writer is instructed to keep the report concise and avoid unnecessary verbosity (an issue noted in early DBOT outputs that the critic later fixed [56] [57]). *Errors:* The Writer might hallucinate or misstate data if the prompt is not constrained – hence the Critic agent (below) will review its output.

- **Critic Agent:** A sub-agent that **reviews and validates** the draft report written by the Writer [47]. The Critic ensures there are *"no loose ends,"* all data in the narrative matches the actual numbers, sources are cited, and the report meets quality and length requirements [47] [58]. Implementation: after the Writer produces a draft, the Critic agent (an LLM prompt) is invoked with instructions like: *"Identify any factual errors, unsupported claims, or sections that are unclear or too verbose in the above report. Provide suggestions for improvements."* The Critic's feedback is then used by either the Supervisor or directly by the Writer (via another LLM call) to revise the report. This could be done in one pass or iteratively until the Critic has no further complaints. The Critic also double-checks that every figure or statement that should have a source indeed has one (e.g. if a growth rate from a 10-K is quoted, a citation to the 10-K or data source is included). **Inputs:** the draft report R (text) and possibly the data I for cross-reference; **Outputs:** either an approval (if all good) or a list of issues/revisions. The Critic agent is crucial for **guardrails** – it acts as an internal auditor ensuring the final output is **trustworthy and correct** [59]. *Errors:* If the Critic itself hallucinates an issue (false positive), the Supervisor may choose to ignore or do a secondary check. *Timeout:* small (a single pass on text is quick for an LLM, e.g. 10 seconds).

Sequence Diagram (Process Flow): The Supervisor orchestrates as follows:

1. **Initialization:** The Supervisor prepares the raw data (DC) for the target company by fetching fundamentals, consensus, and comparables data via APIs. It then calls an LLM (using f<sub>vd</sub>) to map this data to the model inputs I (essentially populating the "Ginzu" spreadsheet inputs) [60] [61]. At this point, the *initial valuation setup* is done, and an initial value v may be computed directly by f<sub>fcf</sub> on I for reference.

2. **Waterfall Phase:** The Supervisor invokes each analysis agent in turn (Valuation, Sensitivity, Consensus, Comparables, News) in a predetermined sequence [22] [62]. After each agent, the central inputs I and valuation v are updated. For example, after the Valuation agent, we have a base valuation; then Sensitivity agent adjusts I (if needed) and we get an array of v's but perhaps keep the original base v for now; Consensus agent might tweak I (say aligning growth to consensus) and recalc v; Comparables agent might not change v but provides context; News agent may update I (e.g. raising cost of capital if news increases risk) and then a new v is computed [29] [63]. The result of the waterfall is a comprehensive initial picture, analogous to Damodaran's typical analysis flow (start with a framing valuation, then question it with market/analysts, comparables, and news) [64] [65].

3. **Iterative Refinement (Router Loop):** After the waterfall, the Supervisor enters a **while-loop** where it defers control to an LLM-based **Router** function to decide the next best action [62] [24]. The prompt

(P<sub>router</sub>) given to the LLM includes the current state (I, v, and possibly a summary of what each agent found) and asks: *“Which agent should be called next to best explain or reduce the gap between DBOT’s valuation and the market price? Provide reasoning.”* [66] [67]. The LLM then responds with a route choice (e.g. “news” if a new headline seems significant, or “sensitivity” if there is still a large discrepancy that could be due to an assumption). Depending on the choice, the Supervisor calls that agent again (with a pertinent prompt/instruction). For instance, if route = “market”, it might mean the system should re-check the market-implied assumptions – effectively calling the Valuation agent with a prompt to align with market multiples (perhaps similar to the initial step) [68] [69]. This loop continues until **convergence**: defined as the point where either (a) the valuation v stabilizes within a certain tolerance over successive iterations, or (b) the router LLM indicates that no further agent calls are needed (it might output a special token or “end” instruction) [70] [71]. In practice, DBOT stops when its estimate and the market price are reasonably reconciled or all avenues have been exhausted [66] [25]. This meta-control where the LLM decides the focus is powerful but also potentially risky – we rely on the LLM’s “general intelligence” to allocate attention appropriately [72] [73]. To maintain determinism and avoid erratic behavior, we use a *consistent prompt and a fixed random seed/temperature* for the router LLM so that given the same state, it makes the same decision (ensuring reproducibility). We also place a hard iteration cap (e.g. max 5 loops) as a safety to prevent infinite cycling.

4. **Report Synthesis:** Once the loop ends, the final I and v are fixed. The Supervisor now composes the full context for the Report Writer: this includes the final spreadsheet inputs and outputs, a bullet list of key findings from each agent (e.g. “Consensus expects lower margins (10% vs our 15%)”, “News: tariff threat could slow sales growth”), and references to images generated. The Writer agent (LLM) is called with prompt P<sub>I⫽v→R</sub> instructing it to produce the valuation report using this information [74] [75]. The output draft R is then passed to the Critic agent for quality check. If the Critic requests changes (which could be things like “too long” or “lacks mention of X risk”), the Supervisor may either adjust the prompt and call the Writer again or directly edit the text (for deterministic outcome) and then finalize it.

5. **Finalization:** The final report (text with embedded charts/tables and citations) is returned as the system output. The Supervisor logs all steps (for traceability in the audit log) and the system resets for the next request.

This architecture is **modular** – each agent can be improved or swapped out independently. For example, we could replace the LLM model behind an agent as better ones become available (the paper notes they used GPT-4 series, but could swap in Claude for the report writing if it’s better at that [76]). Agents communicate via defined interfaces (shared data structures and prompts), making the system akin to a pipeline of microservices managed by the Supervisor.

Agent Interfaces Summary: The table below outlines the interface for each agent in terms of inputs, outputs, and key mechanisms:

Agent	Key Inputs	Key Outputs	Methods/APIs Used	Comments
Supervisor	Cn, Ct, raw DC; intermediate states (I, v); agent feedback	Final report (R); orchestration commands	LLM (router prompt), calls to all other agents, internal logic	Controls flow; no direct external API calls except via sub-agents.
Valuation	I (financials + assumptions); Damodaran methodology context	Updated I (if assumptions tuned); new v	Internal DCF function `f_fcf` (Python); optional LLM for guidance	Performs core valuation math.
Sensitivity	I, current v; target or scenario criteria (optional)	Scenario valuation table; possibly adjusted I	LLM to vary inputs; repeated `f_fcf` calls	Searches for inputs that change v (e.g. to match market).
Consensus	Ct (for lookup); current I (to compare with DBOT's inputs)	Adjusted I (toward consensus); summary of differences	Yahoo Finance via yfinance; Alpha Vantage fundamental API [36] [37]	Brings outside analyst perspective into model.
Comparables	Ct or sector; current valuation multiples in I	Peer list with metrics; relative valuation notes	Yahoo/Alpha Vantage for peer data; EDGAR for fundamentals if needed	Provides context if our valuation diverges from peers.
News	Ct/Cn (search query); current focus (which driver?)	News summary text; suggestions to adjust I (if any)	News API (Google News RSS, etc.); LLM summarization	Captures recent developments affecting the company.
Plotting	I (final numbers; tables); any data series needed	Chart images (file paths or base64)	Matplotlib/Plotly via Python; (LLM-generated code for flexibility) [50]	Visual support for the report (sensitivity chart, comparables chart, etc.).
Writer	Final I and v; all text summaries; chart references	Draft report (Markdown/ HTML)	LLM (GPT-4 or Claude)	Read-only: composes narrative with no data modification [30] .

Agent	Key Inputs	Key Outputs	Methods/APIs Used	Comments
Critic	Draft report; original data (I, v) for cross-check	Feedback on report (or approved final text)	LLM (verification prompt)	Ensures quality, correctness, and adherence to style guidelines.

All agents use the common LLM interface (denoted f<sub>llm</sub>) for their language reasoning needs [77] [78], with specialized prompts P<sub>market</sub>, P<sub>sensitivity</sub>, etc., as described. Non-LLM computations (like f<sub>fcf</sub> for DCF or fetching data via APIs) are encapsulated within the respective agents. Inter-agent data is structured (see Data Model below) to ensure each agent gets the info it needs in a predictable schema.

3. Data Model

We define the key data structures and symbols used throughout the system, consistent with the DBOT algorithm notation [79] [80]:

- **Cn (Company name):** The full name of the target company (e.g. `"BYD Company Ltd."`). Primarily used for display and in news searches.
- **Ct (Ticker symbol):** The stock ticker, used to query financial databases and news (e.g. `"1211.HK"` for BYD's Hong Kong listing, or an appropriate symbol in other markets). Ct is the key for pulling data from APIs (Yahoo, Alpha Vantage, EDGAR) and is passed to the news function *s* [81].
- **DC (Data Collection):** A JSON-like **raw data blob** containing all fundamental data gathered for the company [82]. This includes:

- *Financial statements:* e.g. income statements, balance sheets, cash flows (from EDGAR filings or Yahoo Finance) for recent years or quarters.
- *Key metrics:* e.g. current share price, shares outstanding, market capitalization, debt levels (for WACC calculation), etc.
- *Analyst consensus data:* e.g. projected earnings growth, target price, analyst recommendations (if available).
- *Industry/sector info:* e.g. sector classification, list of competitor tickers.

Example DC Schema:

```
DC = {
  "company": {
    "name": "BYD Company Ltd.",
    "ticker": "1211.HK",
    "exchange": "HKEX",
    "industry": "Auto Manufacturers"
  },
  "market_data": {
    "price": 250.00,
    "market_cap": 72000000000,
    "shares_outstanding": 288000000
  },
  "financials": {
    "income_statement": { "2022": {...}, "2021": {...} },
    "balance_sheet": { "2022": {...}, ... },
    "cash_flow": { "2022": {...}, ... }
  },
  "consensus": {
    "analyst_count": 20,
    "target_price": 300.0,
    "next_year_eps": 10.5,
    "growth_5yr": 0.08
  },
  "comparables": [
    { "ticker": "TSLA", "name": "Tesla Inc", "EV_EBITDA": 16.5, "P_E": 50 },
    { "ticker": "NIO", "name": "Nio Inc", "EV_EBITDA": 12.0, "P_E": null }
    /* ... */
  ]
}
```

This DC is obtained via external data sources. For example, `financials` can be pulled from Yahoo or EDGAR (Alpha Vantage provides endpoints for annual reports in JSON [83]). **Versioning & Provenance:** Each DC item is tagged with source and timestamp metadata (for audit). For instance, `price` might have `"source": "Yahoo Finance API", "as_of": "2025-09-03"`. If multiple sources are used (e.g. one for fundamentals, another for consensus), these are recorded. DC is essentially read-only once fetched; the system may store it as a file or in a cache with a version ID (perhaps the date or a hash of contents) to reuse for repeated valuations or backtesting on that date.

- **I (Inputs for valuation model):** This is the **structured input** to Damodaran's "Ginzu" valuation spreadsheet, derived from DC [84] [60] . It contains the specific parameters needed to compute an intrinsic value. **Key components of I include:**
- **Fundamental current values:** e.g. latest revenue, operating income, tax rate, capital expenditure, etc. (extracted from DC's financials).
- **Value driver assumptions:** The four drivers highlighted are:
    1. **Sales Growth Rates:** e.g. `growth_year1`, `growth_year2_5`, `growth_year6_10`, `terminal_growth`. These might be percentages like 10% for next year tapering to 4% long-term [13] .
    2. **Operating Margin:** current and projected margin (EBIT/Sales). E.g. current 5.9%, next year 5%, then rising to 6.7% mid-term [13] .
    3. **Cost of Capital:** usually the Weighted Average Cost of Capital (WACC) or required return. E.g. 8% based on risk-free rate + equity risk premium + company beta.
    4. **Reinvestment Efficiency:** measured as *Sales-to-Capital ratio*, how much revenue is generated per $1 of reinvestment. E.g. 1.2 in near term improving to 1.6 later [13] . (This inversely relates to how much capital expenditure or working capital is needed for growth.)

- **Forecast horizon structure:** The number of years of detailed forecast vs. terminal value. Typically, Damodaran uses a 5-10 year explicit forecast then a terminal stage. I will include something like `forecast_horizon: 10` years and the yearly or stage-wise assumptions for growth, margin, etc.
- **Macro assumptions:** risk-free interest rate, equity risk premium, tax rate, GDP growth or inflation (if used for context in terminal growth).
- **Calculated interim fields:** The model might carry intermediate results like projected revenues each year, free cash flow each year, present value factors, etc. These can be part of I or computed on the fly. In our implementation, we might not store all intermediate flows in I for brevity, focusing on inputs and final output.

Example I Schema (abridged):

```
I = {
  "company": { "name": "BYD Company Ltd.", "ticker": "1211.HK" },
  "current_financials": {
    "revenue": 420000000000,
    "operating_income": 25000000000,
    "op_margin": 0.059,
    "tax_rate": 0.25,
    "net_debt": 5000000000,
    "shares": 288000000
  },
  "assumptions": {
    "growth_rate": { "year1": 0.10, "years2_5": 0.07, "years6_10": 0.04,
"terminal": 0.025 },
    "operating_margin": { "year1": 0.050, "years2_5": 0.067, "terminal":
0.070 },
    "cost_of_capital": 0.08,
    "reinvestment_rate": { "years1_5": 0.833, "years6_10": 0.625 }
       // Note: reinvestment_rate = 1/(sales-to-capital), e.g. 0.833 corresponds
to 1.2 sales/capital.
  },
  "derived_outputs": {
    "terminal_value": 1.5e+12,
    "intrinsic_value": 420.00,
    "market_price": 250.00
  },
  "analysis_notes": {
    "consensus_adjustment": "Growth lowered from 12% to 10% to align with
analyst view",
    "news_adjustment": "WACC raised 0.5% due to tariff risk",
    "peer_comparison": "BYD EV/EBITDA ~8.8x vs Tesla 17x (cheaper than peers)"
  }
}
```

Here, `current_financials` are from DC. `assumptions` were set by the Valuation agent (or f<sub>vd</sub>) possibly with LLM help, and updated by various agents (consensus, news, etc). `derived_outputs` shows the final result from the DCF model: intrinsic value per share 420 (matching the example) [11], which is included in v (see below). `analysis_notes` is not part of the original algorithm, but we include it to store textual justifications for changes – these can help in report writing and in debugging why certain inputs were chosen.

Versioning: I is essentially *versioned by iteration*. The initial I after mapping DC might be I<sub>0</sub>. After each agent, a new version I<sub>1</sub>, I<sub>2</sub>, ... is produced (though not all agents modify I; some just produce textual output). We track these changes for provenance: e.g. *I.sensitivity_v2* might indicate the version after the sensitivity agent's second iteration. However, to simplify, the Supervisor keeps the current I in memory and logs changes rather than storing multiple copies. The final I is logged as part of the report's appendix or internal record, so that the exact numbers behind the valuation are saved.

- **v (Valuation result):** This represents the **computed value(s)** of the company. In simplest terms, v is the *intrinsic equity value per share* according to the model [85]. It could also represent other measures if needed, but primarily we mean the fair stock price. In the algorithm, v is updated whenever f<sub>fcf</sub> is applied to I (e.g., after each agent's adjustments) [28] [86]. We maintain v as a numeric value (float). If needed, we can enrich v to a structure that includes components like enterprise value or a range. For example, after sensitivity analysis, v might be a range or distribution rather than a point; however, we can keep the "current best estimate" in v, and have the sensitivity results in I or a separate structure.

Example: `v = 420.00` (meaning fair price $420.00/share). If we consider more, we might have `v_high=468, v_low=417` from the sensitivity table [87], but those can be stored in I's analysis notes or separate.

We will output v in the report compared to the actual market price to make the recommendation (if v >> market price, it's a "buy/undervalued" signal, etc.). Provenance of v is inherently tied to I's provenance, since v = f<sub>fcf</sub>(I). We ensure to log which version of f<sub>fcf</sub> (if the model is updated) was used, to maintain consistency (for example, if we update the discount rate formula in a future version, it's noted). For MVP, f<sub>fcf</sub> is fixed as Damodaran's standard DCF.

- **f<sub>fcf</sub> (Free Cash Flow valuation function):** This is not a data object but a function mapping I → v [88]. It encapsulates the DCF calculation: projecting free cash flows from the inputs, discounting them to present, and outputting an equity value. It uses the assumptions in I such as growth and margins to forecast revenues and profits, applies reinvestment to compute cash flows, and discounts at the cost of capital. Taxes and other factors are applied as per standard valuation. We implement f<sub>fcf</sub> in Python (e.g. using Pandas or Numpy for calculations). It's effectively our **valuation engine** (detailed in Section 5).

- **s (News search function):** A function s: Ct → text, which given a ticker returns aggregated news text [89]. This can be a wrapper around an API call or web scrape. For example, s(Ct="1211.HK") might query Google News for "BYD Co Ltd" and return the top N headlines and articles concatenated. This raw text is then processed by the News agent's LLM prompt (f<sub>Dnews</sub> in the paper) to update I or produce a summary [90]. In implementation, s is separate from the LLM; it's more of a data fetcher.

- **f<sub>llm</sub> (LLM function):** This denotes the general interface to an LLM, e.g. GPT-4, that all agents use for reasoning tasks [81] [91]. It can be thought of as f<sub>llm</sub>(prompt ⫽ data) → text. We use specialized sub-functions of f<sub>llm</sub> for each agent prompt as described in Algorithm 1 [77] [78]:

- f<sub>vd</sub>: maps raw Data to inputs I (the initial setup) [60].
- f<sub>valuation</sub>: given a prompt (like P<sub>market</sub> or others) and I, returns modified I (this encapsulates the Valuation agent's LLM reasoning if any) [45] [92].
- f<sub>Dnews</sub>: given prompt and current I, plus fetched news text, returns an updated I (incorporating news insights into assumptions) [90].
- f<sub>news_summary</sub>: summarizes raw news text into concise form [44].
- f<sub>plot</sub>: as mentioned, takes prompt and I to produce plotting code, which then yields an image [50].
- f<sub>report</sub>: generates the final textual report from prompt, I and v [53].
- The **Router** uses f<sub>llm</sub> with P<sub>router</sub> to decide next step [68].

Each of these calls uses the same underlying LLM API but with different instructions and data. In practice, we will implement a module to manage LLM prompts and responses, including handling of API keys, rate limits, and caching of responses (to ensure repeatability where needed).

File Schemas & Storage: The system will manage a few file types: - **Configuration files:** e.g. a YAML/JSON for API keys (Alpha Vantage, OpenAI) and global settings (like default risk-free rate, etc.). - **Data cache files:** Storing DC for companies (could be JSON files named by ticker and date). Also news articles could be cached as text files by date. - **Model input/output logs:** We may store the final I and v for each run (e.g. as JSON, or embed in the report as an appendix). This serves as a record for backtesting and auditing. Versioning of these files might include the date and a git-like hash of content for integrity. - **Report output:** likely stored/exported as Markdown or PDF. The report contains the narrative and references to charts (which might be separate image files in an output directory). - **Agent conversation logs:** We maintain logs of LLM prompts and responses for debugging and governance. Each interaction can be stored (with sanitized content if needed for privacy) in a log file or database with timestamps.

Provenance is critical – for any number in the final report, we should be able to trace it back to either the original DC data (with a source citation) or an intermediate computation. Our data model supports this by keeping `analysis_notes` in I, as well as explicit fields for source of each data point in DC. Additionally, the Critic ensures all such references are cited in text, creating a human-readable provenance in the report itself [93]. For internal use, we could tag each number in I with a source pointer (e.g. `revenue.source = "10-K 2022, EDGAR file XYZ"`). This level of detail might be added in future iterations as needed for full audit trails.

4. Orchestration and Control

The **orchestration logic** is implemented by the Supervisor agent, following a two-stage control pattern: a fixed **waterfall sequence** followed by a flexible **iterative loop** [22] [62]. The design ensures that control decisions are either deterministic or reproducible, to make the system's behavior predictable and testable (an important requirement for trust [10]).

Waterfall Phase: On each new company analysis, the Supervisor initiates a top-down execution of agents in a predetermined order: 1. **Initial Data Mapping:** Using f<sub>vd</sub>, convert raw data DC to initial valuation inputs I [60]. This step is akin to filling in the Damodaran spreadsheet with the latest financials and some baseline assumptions. For MVP, this can be a simple deterministic mapping (e.g. take last year's revenue growth as initial forecast, etc.), possibly refined by an LLM prompt that infers reasonable starting assumptions from the data. 2. **Initial Valuation:** Compute an initial v = f<sub>fcf</sub>(I) as a base case. This could be considered the **"market" step** if we incorporate the current market price as a reference. In the algorithm, the first waterfall call is labeled "Market valuation" [28]. We interpret this as asking the LLM/agent to consider the market's perspective – perhaps by comparing v to the market price and adjusting if needed. In practice, we might skip directly to step (3) or treat this initial f<sub>fcf</sub> as the "as-is" valuation (which likely will differ from market). 3. **Sensitivity Analysis:** Call the Sensitivity agent (P<sub>sensitivity</sub>) to test key assumptions [86]. This might not change I permanently, but produces a table of alternative outcomes. However, if, for example, the market price is far from v, the sensitivity agent could identify a scenario (like "if growth = X%, then valuation = market price") and we might choose to update I toward that scenario for further analysis. For now, we run it and keep results in memory. 4. **Consensus Incorporation:** Call Consensus agent (P<sub>consensus</sub>) to pull analyst expectations and adjust I accordingly [94]. E.g., if analysts universally predict lower margins, update the margin assumption in I. Then recalc v via f<sub>fcf</sub>. The new v might move closer to market or not. 5. **Comparables Analysis:** Call Comparables agent (P<sub>comparables</sub>) [95]. Provide it the current I and v, and any peer data from DC. It returns context on relative valuation. Usually, we won't alter I here (unless perhaps we find we've been too optimistic/pessimistic after seeing peers – an advanced step could adjust, but that might be left for the loop). We still recalc f<sub>fcf</sub>(I) (which likely unchanged if I unchanged) just to maintain sequence. 6. **News Analysis:** Call News agent last in the waterfall (f<sub>Dnews</sub>) [63]. Provide it the ticker; it fetches news (s(Ct)) and integrates any important updates into I [63]. For example, it might increase the risk premium if news indicates higher risk, thereby increasing cost of capital in I. After this, we run f<sub>fcf</sub> one more time to get the post-news valuation v.

At the end of the waterfall, we have an initial report outline in terms of data: a possibly modified I and a valuation v that has considered multiple angles, plus collected outputs like the sensitivity table, news summary, etc. This design mirrors Damodaran's process: start with a story and base numbers, then **"question the numbers"** using market, analyst, and comparative information [64].

Determinism in Waterfall: The waterfall sequence is mostly deterministic given the input data, aside from any randomness in LLM outputs. To minimize variability, we use the LLM in a mostly analytical way: e.g. for consensus and comparables, the LLM is primarily formatting or minor reasoning, so we set a low temperature (close to 0) to get consistent outputs. The initial f<sub>vd</sub> mapping might have some variance if fully done by LLM; to control that, we may implement f<sub>vd</sub> as a deterministic procedure or a very constrained prompt (like "extract last year's revenue growth and set that as year1 growth" which has a clear answer). By doing so, two runs on the same data should yield the same initial I and thus the same waterfall trajectory.

Contextual Iterative Loop: After the waterfall, DBOT might still have a gap between its valuation v and the actual market price, or unanswered questions about the valuation. The iterative loop lets the system dynamically decide what to explore further: - The Supervisor prepares a **Router prompt** P<sub>router</sub> that encapsulates the situation: it might say (in pseudo-language): *"DBOT's current valuation is \$420 vs market price \$250 (68% higher). Agents have provided their inputs: (Consensus: market expects lower growth, News: tariff risk etc., Comparables: BYD cheaper than peers…). What is the most useful next step to explain or* *reconcile this difference? Options: {market, sensitivity, consensus, comparables, news, end}. Reply with the option and an instruction."* This leverages the LLM's ability to interpret the context and choose an action [66] [67]. - The LLM's response (f<sub>llm</sub>(P<sub>router</sub> ⫽ I ⫽ summary)) will contain either one of the agent names or "end". If it chooses an agent, it may also provide a brief **instruction** or goal for that agent (the algorithm notes `route, instruction ← f_llm(P_router ⫽ I ⫽ v)` [68]). For example, it might output: *"sensitivity: Try lowering the growth further to see if value aligns with market"* or *"news: Check if there are any recent developments on regulatory actions that could justify the market's pessimism."* - The Supervisor then calls the chosen agent with that instruction. If "sensitivity" was chosen with instruction, we feed that as the new prompt to Sensitivity agent (e.g. P<sub>sensitivity</sub> = *"Find what assumptions (perhaps growth) would make the intrinsic value ~$250."*). The agent executes, updates I or generates results, and we then recalc v. - After the agent returns, the Supervisor may incorporate the new findings (for instance, if sensitivity found that a 4% growth yields $250 value, maybe now we know the market is pricing in much slower growth). The context (I, v, and perhaps an updated narrative state) is updated. - The loop then repeats: call the router LLM again with the new state. Over successive iterations, DBOT will explore various "avenues" until it either **converges** or the LLM decides to **break** out [70] [71]. Convergence is detected if the LLM returns "end" or if v doesn't change significantly for a couple of iterations (we can set a threshold like <1% change in v). - The idea is DBOT allocates attention like a human analyst might: if after all that the reason for divergence still isn't clear, perhaps re-check initial assumptions ("market" route) or if new news came during analysis, re-check news. This **context-driven control** is a novel aspect, essentially letting the LLM meta-control its workflow [72] [96].

Determinism and Safety in Loop: While the LLM router adds adaptivity, it could introduce nondeterminism. To mitigate that: - We use a **fixed random seed or low temperature** for the router LLM. The prompt itself is relatively straightforward (select from known options), so a greedy or low-temperature decoding should consistently pick the same path given identical input. - We implement a maximum number of iterations (say 5). If reached, Supervisor will break the loop to avoid endless cycling. - We monitor for oscillation (if it goes, say, consensus -> sensitivity -> consensus -> sensitivity repeatedly). If detected, we break and potentially let the Critic note that DBOT couldn't fully reconcile some aspects, which might be a point for human review. - The final decision to end can be made by either the LLM explicitly choosing "end" or by the Supervisor after some stable iterations. The algorithm allows either (the else branch on route handles unknown or end commands) [71].

Error Handling: The orchestrator must handle any agent failures gracefully: - If a data fetch agent fails (e.g., API down), the Supervisor can skip that agent or use cached data. For instance, if the News agent fails to retrieve news, log a warning and continue without news. - If the LLM for a critical step (like Writer) fails or returns invalid output, the Supervisor can retry or fallback. We plan simple retry logic for LLM calls (retry once on failure or empty response). - If the router LLM returns an invalid route (not one of expected) or gibberish, we break the loop (as per algorithm's else) [71] and proceed to finalization. The Critic can then note a limitation if needed.

Control State Machine Pseudocode: Below is pseudocode illustrating the orchestrator logic, combining waterfall and loop:

```
# Initialization
DC = fetch_data(company_ticker)
I = map_data_to_inputs(DC)         # via f_vd (LLM or deterministic)
```

```
v = f_fcf(I)                       # initial valuation

# Waterfall sequence
for step in [P_market, P_sensitivity, P_consensus, P_comparables, P_news]:
    if step is P_news:
        # News agent: incorporate news into I, then value
        news_text = fetch_news(Ct)
        I = f_Dnews(P_news ⫽ I ⫽ v ⫽ news_text)   # update inputs with news
impact
        v = f_fcf(I)
    else:
        # Other agents: use f_valuation with respective prompt
        I = f_valuation(step ⫽ I)   # LLM may adjust I based on step logic
        v = f_fcf(I)                # recalc valuation after adjustment

# Iterative refinement loop
loop_count = 0
max_loops = 5
last_v = v
while loop_count < max_loops:
    route, instr = f_llm(P_router ⫽ I ⫽ v)   # LLM decides next focus
    if route in ["market", "sensitivity", "consensus", "comparables", "news"]:
        switch route:
            case "market":
                I = f_valuation(P_market ⫽ I)      # revisit market-related
adjustments
            case "sensitivity":
                I = f_valuation(P_sensitivity ⫽ I ⫽ instr)  # use instruction
in prompt
            case "consensus":
                I = f_valuation(P_consensus ⫽ I)
            case "comparables":
                I = f_valuation(P_comparables ⫽ I)
            case "news":
                news_text = fetch_news(Ct)
                I = f_Dnews(P_news ⫽ I ⫽ v ⫽ news_text)
        v = f_fcf(I)
        loop_count += 1
        if |v - last_v| < convergence_threshold:
            # If change in v is negligible, consider converged
            break
        last_v = v
        continue   # next iteration
    else:
        break  # LLM chose "end" or unrecognized route
end while
```

```
# Final report generation
final_report = f_report(P_Iv_to_R ⫽ I ⫽ v)   # Writer produces report draft
critic_feedback = f_llm(P_critic ⫽ final_report ⫽ I)  # Critic reviews the
draft
if critic_feedback indicates issues:
    # Simple approach: include feedback in prompt and regenerate or fix
iteratively
    final_report = f_report(P_Iv_to_R_adjusted ⫽ I ⫽ v ⫽ critic_feedback)
    # (In practice, might loop until satisfied or fix minor issues by code)

return final_report
```

This pseudocode aligns with Algorithm 1 in Appendix A of the paper [97] [98] [99], with slight adjustments for clarity. Each agent's invocation either updates I or not, but we always recompute v = f<sub>fcf</sub>(I) after an agent that could affect value, ensuring v is up-to-date for the next decision.

Determinism vs Adaptivity: We achieve a balance by making the waterfall fully repeatable given the same data, and confining any nondeterministic reasoning to the contextual loop where it adds value. Even there, with controlled random settings, two runs on the same data should choose the same sequence of routes (the loop essentially becomes deterministic if the LLM is deterministic). This is crucial for backtesting – if we rerun DBOT on historical data, we want the same outputs each time (otherwise performance evaluation is impossible) [10]. Logging every decision ensures we can trace why a particular route was chosen (the LLM's reasoning string can be saved, even if it's just for debugging insight).

Convergence Criteria: We define convergence in practice as either no significant changes in v or no new information being added. For example, if DBOT has iterated through each agent and the gap remains because of something outside its current knowledge, further iteration may not help. At that point, it stops and acknowledges the gap in the report (which a human might interpret as market pricing in something intangible like sentiment). This prevents overfitting to force a match with market – sometimes a difference will remain, and that itself is a finding (e.g., "DBOT believes the market is underestimating the company, hence sees it as undervalued").

Summary: The orchestration logic ensures a **structured yet flexible** analysis. The initial pass covers all bases systematically, and the loop then zeroes in on the most pertinent factors causing deviation between DBOT's valuation and reality [66] [100]. This yields a report that not only states a valuation, but explains *why* it differs from market consensus, exploring multiple angles – a key to mimicking a thorough human analyst. Moreover, by logging each step and maintaining determinism, we preserve an audit trail and enable scientific evaluation of DBOT's performance over time.

5. Valuation Engine

The core of DBOT is the valuation engine grounded in a **free cash flow (FCF) Discounted Cash Flow model** [101] [7]. This engine takes the inputs I (as defined in Section 3) and computes an intrinsic value per share (v) for the company. It is essentially an implementation of Damodaran's valuation approach ("Ginzu" spreadsheet), which is based on projecting cash flows from the company's **story (assumptions) to numbers** and discounting them [30].

Model Specification: At a high level, the model does the following: - **Project financials over a explicit forecast period** (e.g. 10 years): - Start with current revenues (from I.current_financials). - Apply the **sales growth rate** assumptions: e.g. if revenue is $100 in year0 and growth year1 = 10%, year1 revenue = $110; for years 2-5 at 7%, it grows accordingly each year, etc. - Apply **operating margin** assumptions to get operating profit (EBIT) each year. E.g. year1 margin 5% on $110 revenue gives EBIT $5.5. - Calculate **tax** on EBIT (using either a fixed tax rate or something from I, say 25%). - Result: **Net Operating Profit After Tax (NOPAT)** each year. - Determine **reinvestment** each year needed to drive the growth. This is where the *reinvestment efficiency (sales-to-capital ratio)* comes in. If the ratio is 1.2 (as in BYD example, meaning $1 of capital yields $1.2 sales), the *reinvestment rate* (as fraction of NOPAT reinvested) is roughly `growth / return_on_capital`. Alternatively, using the ratio directly: change in sales / sales_to_capital = capital invested. For instance, if sales grow from $100 to $110 (change $10) and sales-to-capital is 1.2, then required new capital is $10/1.2 ≈ $8.33. If after tax operating earnings margin is stable, this capital comes out of cash flows. Essentially, **FCF = NOPAT - reinvestment**. The model likely uses formulas: reinvestment = (growth_rate / sales_to_capital) * sales, or something similar, ensuring internally consistent growth. - We may also incorporate **depreciation** and other non-cash charges if needed, but Damodaran's approach often simplifies to focusing on NOPAT and net reinvestment (capex - depreciation + change in working capital). Our inputs might implicitly cover that by the reinvestment ratio. - Compute **Free Cash Flow to Firm (FCFF)** each year = NOPAT - reinvestment. - **Compute terminal value** at end of forecast period: - For year 10 to perpetuity, use terminal growth (g<sub>terminal</sub>) and the stable period assumptions (margin, reinvestment, cost of capital presumably also stabilized). - Terminal Year FCF = year10 NOPAT * (1 - g/ROIC) or simply year10 FCF grown by terminal growth if assuming stable reinvestment yields that same growth. - Terminal Value = FCF<sub>terminal_year+1</sub> / (WACC - g<sub>terminal</sub>), the standard perpetuity formula. - If the company has significant debt, we might compute enterprise value (PV of FCFF) then subtract debt/add cash to get equity value. Alternatively, one can directly compute equity cash flows, but it's likely they do FCFF and then adjust for debt. - In our I, if net_debt is provided, we will subtract net debt at the end to get equity value. Also if any non-operating assets (excess cash, holdings), those would be added – though not explicitly in our schema, but worth noting as future expansion. - **Discount all cash flows to present value** using the **cost of capital (WACC)**: - We have WACC given in I (cost_of_capital). We assume it's nominal and appropriate for the currency of cash flows. - Each year's FCF PV = FCF<sub>t</sub> / (1+WACC)^t. - Sum PV of years 1-10 plus PV of Terminal Value = Enterprise Value. - If using FCFE (free cash flow to equity), we'd discount at cost of equity and not subtract debt, but given the context we use FCFF and WACC in a standard way [101] [7]. - **Output the intrinsic equity value per share (v)**: - Equity Value = Enterprise Value - Net Debt (from I.current_financials). - v = Equity Value / shares_outstanding.

We will implement this in Python (perhaps as part of the Supervisor or as a standalone function the Valuation agent calls). The structure is straightforward to code with loops or vectorized operations for forecast.

Drivers and Term Structure: The four value drivers (growth, margin, reinvestment efficiency, cost of capital) are explicitly part of I. Each may have multiple phases: - Growth: we allow a multi-stage (could define up to 3 stages: short, medium, long/terminal). - Margin: similarly can converge from current to a target. - Reinvestment/Sales-to-capital: can change over stages as company matures (as in BYD example, reinvestment efficiency improves in later years [13]). - WACC: often we might keep one constant WACC. Possibly adjusting it if capital structure changes, but probably fixed or a simple linear change to a stable figure.

The model can handle these term structures piecewise. E.g., years 1-5 use one set, years 6-10 another, and terminal after that.

Taxes: The model uses a tax rate (either current effective tax or marginal tax rate) to calculate after-tax operating profit. Damodaran often uses a marginal tax rate in forecasts (to normalize future taxes). We include a tax_rate in I.current_financials. For simplicity, use that constant rate on EBIT for all future periods.

Discounting and Present Value: All cash flows are in nominal terms if we use nominal WACC. We should ensure consistency: if using nominal rates, growth includes inflation. If needed, macro inputs like expected inflation could be part of assumptions to ensure, say, long-term growth doesn't exceed nominal GDP growth. But we can rely on user/agent judgment for picking a terminal growth (often a bit below GDP).

Macro Inputs: Some macro factors indirectly influence the valuation: - **Risk-free rate:** part of cost of capital (WACC = Rf + equity risk premium*beta etc.). We may let the system fetch current 10-year treasury yield as Rf from an API or have it configured. For MVP, possibly a fixed 4% or so can be set, or manually input per market. - Equity Risk Premium & Beta: not explicitly in I, but underlying WACC and cost of equity if we break it down. We might allow I to include beta and ERP for transparency. However, since we give WACC directly, those components might not be needed unless the model is extended. - Country risk, FX rates: for multinational companies, cost of capital might include country risk premiums. Damodaran often adjusts for country risk in discount rates. For now, we may not delve into that – but if needed, the News agent or consensus could adjust WACC for risk (e.g. raising WACC if political risk is noted). - Economic indicators:** if something like current risk-free or GDP growth needed, we can fetch it. Alpha Vantage or other free sources have treasury yields or we could hardcode an approximate. Not central to initial prototype, but to mimic Damodaran's practice, we at least allow setting Rf and ERP.

Edge cases: For some companies (financial firms, startups with negative cash flows, etc.) the model needs tweaks: - Financials (banks) are valued differently (dividends or book value models) – DBOT likely excludes them or uses different logic. In our design, we assume standard non-financial companies. - High-growth startups might have negative cash flows early. The model can handle that as long as growth and margin eventually lead to positive cash flow in terminal period. Our agents (especially f<sub>vd</sub>) should make reasonable assumptions to ensure the company turns profitable by terminal (or else DCF might undervalue a perpetually loss-making firm). - If cost of capital <= growth (which should not happen in terminal by assumption), the formula would break (denominator zero or negative). Our Critic or guardrails should ensure terminal g < WACC to avoid that error (if an agent sets terminal g too high, Critic flags it or we clamp it).

Validation of Model Output: To ensure the engine works correctly: - We will test it on known examples (like some Damodaran blog model outputs if available, or at least consistency: if we input same assumptions as Damodaran did for BYD, do we get ~$420). - The Critic agent can cross-verify certain simple checks: e.g., if growth is positive, ensure reinvestment is also positive; if we increased cost of capital, final value should not increase, etc. These sanity checks can catch mis-implementation or input errors. - Also, for debugging, we can output intermediate calculations (like a mini table of Year 1-10 FCF) not in the report but in logs or for the Critic to see if needed. However, that might be advanced.

Integration with Agents: The Valuation agent primarily calls this engine. Other agents indirectly interact: - Sensitivity agent might call the engine many times with different I variants. - Comparables agent might use the output (intrinsic value vs market to compute a P/Intrinsic ratio or so). - Writer uses final outputs for narrative. - Critic could recalc the model independently to verify (maybe an idea: have a simpler code version to double-check the LLM didn't fudge anything, but since we trust our code more, it's fine).

FCF Model Example: Using the values from BYD example summary [13] : - Year0 revenue (2024) is implicit. Growth 10% → Year1 revenue gets 10%, margins drop to 5.0% giving NOPAT; reinvestment efficiency 1.2 yields some reinvestment; FCF year1 computed. - We would actually need actual numbers to reproduce $420, but since they gave final output, we trust it matches their numbers. - Terminal EV/EBITDA multiple of 8.8x mentioned [14] , which our model should reflect: If BYD's terminal EBITDA and EV from model yields ~8.8, that's a consistency check (the model likely implied that multiple). - Our design can optionally compute such implied multiples from model outputs to mention in reports.

Conclusion: The valuation engine is the quantitative backbone. It **implements a classic DCF** with company-specific and macro inputs, producing an intrinsic value that the rest of the system uses as a reference. By keeping this engine in Python code (using libraries for accuracy), we ensure the numeric part of DBOT is **stable and free from LLM errors**, aligning with the goal of credible, backtestable valuations [10] . The engine is also **transparent**: each input's effect on output can be traced (e.g. via the sensitivity agent or by printing the cash flow forecast), reinforcing trust in the system [102] [103] .

6. News Pipeline

The News Pipeline is responsible for injecting **current events and qualitative context** into the valuation process in a controlled manner. It emulates how a human analyst reads news and assesses which events materially affect the company's valuation drivers [104] [105] .

Sources and Fetching: We rely only on **open/free news sources**: - Use a news API or search (e.g. *Google News API*, or Bing News Search) to fetch recent articles for the company. Query by ticker and company name (to ensure broad coverage). For example, search "BYD Company news last week". If using Google News RSS (which is open), we can retrieve a feed of recent headlines. - We target reputable publications (the paper mentions Wall Street Journal, Financial Times, etc. as sources [106]). While those specific sites often have paywalls, their headlines are accessible and sometimes snippet via Google. We might augment with general news sites or press releases. - Limit to, say, the last **30 days** of news to keep it relevant and not overload the LLM with too much text.

De-duplication: News APIs often return duplicate or very similar stories (e.g., many outlets reporting the same earnings results). The pipeline will: - Compare headlines; remove those that are essentially the same or from the same event. - Perhaps prefer the more authoritative source if duplicates exist (e.g., keep WSJ article, drop others on the same topic). - Ensure we have a diverse set of news covering different aspects (market news vs. industry news vs. macro news affecting the company).

Three-Phase Reading (Emulation of Human Reading): As described [41] [107] , the News agent goes through phases: 1. **Headlines only:** The agent first skims just the titles of, say, the top 10 news items. Using an LLM (with a prompt like: *"Given these headlines, which seem relevant to the company's fundamentals or could impact its valuation drivers?"*), it picks which ones merit deeper reading and what it guesses the impact could be from headline alone. For example, a headline "BYD to Double Battery Production in 2025" clearly signals something about growth potential – the agent would mark this as relevant to *sales growth* driver. 2. **Headline + Lead Paragraph:** For the subset of relevant articles, fetch the first paragraph or a summary. The agent (LLM) now assesses the content: *“Does the first paragraph confirm a significant development? How might this affect sales, margins, risk, etc.?”*. It might note specific facts (e.g. “BYD’s EV sales rose 50% in Q3”) and early insights (like “EU tariff on Chinese EVs” could mean a risk to margins in Europe). 3. **Full Article Read:** For the top 1-3 articles that seem *most important* after step 2, the agent fetches the full text (if accessible) and reads it thoroughly. Then it extracts the key points relevant to valuation. It might use a prompt: *“Summarize the article focusing on any information that changes the outlook for the company’s cash flows or risk.”* This could yield, say: *“Article: EU considering 10% tariff on Chinese EVs -> could reduce BYD’s European sales growth and margins (News, Sep 2024).”*

After these phases, the agent has gathered a set of *material news points*. It then produces a **final news summary** that consolidates them [39] [46]. This summary is structured (for the report) often as a short section highlighting major news and the implied effect: - E.g. "BYD’s EV deliveries are rising steadily (except a Q1’24 dip due to EU tariffs dispute) [12]. Global EV sales projected to grow from 8M to 60M by 2040, supporting BYD’s growth [12]. However, looming trade-war threats (e.g. potential US tariffs on Chinese goods) introduce caution [108] [109]."

Factuality and Verification: We must ensure the news summary is factual: - Use multiple sources to cross-verify big claims. If one article says "BYD’s margin hit 10% record," ideally confirm via another or via the actual financials (if it contradicts our DC data, trust the actual filings). - The Critic agent will check that any specific figures or claims from news that end up in the report are supported. For instance, if we state EV sales rising to 60M by 2040 [110], the Critic will ensure a source is cited (the paper’s BYD example likely took that from a projection source). - We include citations for news in the report wherever possible (either citing the news source or at least stating "according to [Source]").

We might implement a simple check: when the News agent produces its summary, have the Critic or a small function scan for numbers in the text and see if those numbers can be found in DC or other sources. If the news says "8.9x TEV/EBITDA", we might verify that with our comparables data in DC (if available) or trust it if it's an analysis in the article. Since our aim is to stick to facts, we likely trust known sources like FT, but still.

Mapping News to Value Drivers: One innovative part is that the agent doesn’t just summarize but connects news to valuation *drivers*. We design the LLM prompt to explicitly do this: e.g., *“For each important news item, identify which of {sales growth, operating margin, cost of capital, reinvestment efficiency} it impacts and in what direction.”* This way, the News agent might produce output like: - "News: New battery tech announced -> could improve operating margin (lower costs)". - "News: Trade war concern -> increases cost of capital due to higher equity risk premium for China". - "News: Government EV subsidies -> boosts sales growth potential."

The agent can then take immediate action by adjusting I or flagging that a change might be needed: - If something clearly warrants a change, we could have the News agent call f<sub>Dnews</sub> to update I (the algorithm suggests f<sub>Dnews</sub> incorporates news into I and then recalculates v in the waterfall [63]). For instance, if cost of capital should be +0.5%, the agent does that. - If it’s more speculative, the agent might not change I, but note the risk. For instance, "tariffs could lower growth, but not certain yet" – might not adjust the base case but include narrative around it. In the BYD example, after news, DBOT didn’t change the $420 value drastically, but it noted caution in narrative [111] [109].

Iteration and Timeliness: The News pipeline can be invoked again in the iterative loop if needed: - If after initial analysis the gap between DBOT and market is large, maybe DBOT thinks "perhaps there’s some news

I missed" – the router might call news again. In such case, we could widen the search (maybe older news or niche sources). - But we need to avoid chasing too much news in loops (since new news doesn't appear in minutes usually). So maybe limit news agent to run at most once per analysis or once per day.

Example (BYD news from Appendix B): The example mentioned: - Tariffs from the EU and disputes with foreign regulators caused a blip in early 2024 deliveries [12] – that's a news-derived insight impacting sales (short-term dip). - Threats of US tariffs (in context of a potential Trump win) influencing a tone of caution in later reports [108] – that's news affecting risk/cost of capital or at least perceived risk. - These were integrated into DBOT's narrative and possibly its assumptions (maybe DBOT kept growth but flagged risk, or reduced growth slightly).

Our pipeline would capture similar items: - If such news is present, note it and either adjust growth downward for Europe segment or adjust WACC upward for geopolitical risk. For now, maybe the agent just flags it, and the decision to adjust is left to the loop (the router might later say "maybe revisit market with lower growth if news says adoption slower").

Summary: The News pipeline brings **real-world dynamics** into the valuation: - It ensures the final valuation isn't stale or purely numbers from last quarter – it reflects things like technological breakthroughs, regulatory changes, competitive moves, etc. - It does so in a targeted way, extracting only what matters for fundamental value (filtering out general market noise) [107]. - By structuring news analysis in phases and linking to drivers, it maintains a logical connection between qualitative events and quantitative assumptions, which strengthens the report's explanations.

The result is that DBOT's analysis will cite current events as rationales for its adjustments or its confidence. E.g., *"BYD's improved battery cost will likely boost margins (news from Oct 2024), contributing to our valuation uptick"*, or *"However, political risks (e.g. tariff threats) justify a higher discount rate"*. This mimics human analysts who often cite news and developments in their reports.

7. Consensus and Comparables

This section deals with how DBOT leverages **external viewpoints** – both the collective view of market analysts (consensus) and the valuation of similar companies (comparables) – to challenge and refine its own valuation.

Analyst Consensus Integration: Analysts' forecasts and opinions can provide a check against DBOT's model assumptions: - The Consensus agent fetches data like **mean projected earnings growth**, **revenue forecasts**, **target price**, and sometimes **analyst recommendations** from open sources. Yahoo Finance, for example, provides analyst estimate summaries (e.g., next year earnings, 5-year growth rate) and a consensus target price for many stocks. - These data points are then compared to DBOT's assumptions in I. For example: - If DBOT assumes 10% revenue growth but consensus 5-year growth is only 8%, DBOT might reconsider if it's too optimistic. - If analysts' average target price is much lower than DBOT's v, that signals DBOT might be overlooking something or just that analysts are more bearish – either way it should be addressed. - The agent could adjust inputs or at least annotate differences. For instance, DBOT might decide to temper its growth assumption slightly towards consensus *if* it doesn't have strong reasons not to. Alternatively, it might keep its higher growth but then explicitly state "our growth forecast is higher than consensus due to X reason" – which the Writer can include. - Another consensus element is **earnings or sales surprises**. If recently actual earnings diverged from consensus, analysts might revise forecasts.

DBOT's data (DC) includes actuals, so differences to consensus forecasts can highlight trends (analysts overly pessimistic or optimistic historically). - **Bias handling:** Analysts can be systematically biased (often overly optimistic on long-term growth or slow to adjust). DBOT being systematic might account for this by not following consensus blindly. Perhaps the Critic ensures that DBOT doesn't just match consensus to avoid a bias of crowd-think. However, measuring or correcting bias is complex; for now, DBOT might just present consensus as an input and maybe weight it. In backtests, one could analyze if DBOT's deviations from consensus correlate with results, but that's beyond initial scope.

In implementation, after fetching consensus: - We create a small structured data (like `consensus.next5yr_growth = 0.08`). - We feed a prompt to LLM: *"Analysts expect ~8% growth vs our 10%. Should we adjust our model or explain this difference?"* We likely have the LLM decide minor tweaks to I or decide to keep differences but note them. - The algorithm's pseudocode shows consensus as one of the steps in waterfall: f<sub>valuation</sub>(P<sub>consensus</sub> ∥ I) then f<sub>fcf</sub> [95]. So presumably, that prompt might say "Incorporate analysts' estimates into the model." A reasonable outcome: DBOT might produce an alternate valuation using analyst assumptions, possibly to compare to its own. - Perhaps the agent might output a separate valuation (v_consensus) based on consensus inputs (like plug consensus growth and run f<sub>fcf</sub>). Then the report can say "If we use analyst consensus growth of 8%, our valuation would be $X (close to or far from our base $Y)". This was not explicitly in text, but it's a possible approach.

Comparables (Relative Valuation) Logic: The Comparables agent provides a **market-relative perspective**: - Identify **comparable companies**. Typically those in the same industry and similar size or business model. For example, BYD's peers included Tesla, Nio, XPeng, Li Auto in the EV space [14]. Data sources: Yahoo Finance's summary often lists "Peers" (e.g., for BYD it might list Tesla, Nio, etc.). We can also use industry classification (like all Auto Manufacturers in China or globally). - For each peer, gather key multiples and metrics: P/E ratio, EV/EBITDA, revenue growth, margins, etc. Many of these can be scraped from finance sites or via APIs: - Yahoo's `Ticker.info` or `yfinance` might give some multiples for company (though sometimes only for the target, not peers). - Alternatively, use an API like Financial Modeling Prep or simply a quick web scraping of a site like FinViz or Reuters that lists peers. - We likely focus on one or two telling metrics, such as EV/EBITDA (as the paper's example did) or P/E, or Price/Sales if earnings are negative. - Compare the target's metrics to peers: - If DBOT's company has a significantly lower multiple (like BYD's 8.8x EV/EBITDA vs Tesla 17x [14]), it suggests undervaluation (assuming growth prospects are not worse). - The agent should adjust for differences: e.g., if our company has lower margin or growth than peers, a lower multiple might be justified. - An LLM can help here by reading the peer data and reasoning: *"BYD's EV/EBITDA is ~8.8x, roughly half of Tesla's. Given BYD's growth is strong and comparable to peers, this indicates BYD might be undervalued by the market relative to others"* [14]. - The Comparables agent may provide: - A **chart or table** of the multiples (Plotting agent will use this). - A suggestion whether DBOT's valuation should be adjusted. If comparables uniformly trade higher, maybe DBOT's valuation (based on fundamentals) already reflects that gap, or if not, DBOT might question if it missed some risk factor (maybe peers are overvalued or our target has hidden issues). - Possibly in the loop, if DBOT's value is way above market but comparables are also priced high, DBOT might realize the entire sector is priced low (maybe due to macro pessimism) and might not adjust its number but will highlight the relative case (a potential "value play" argument). - **Peer logic and symbol mapping:** This can get tricky if the company is conglomerate or unique, but for most, we can stick to industry. For mapping: - Yahoo's API could give sector & industry classification for Ct. We then search our internal mapping or external API for top companies in that industry by market cap or region. - Alternatively, if consensus data provides a list of "comparables analysts used", that would be ideal, but not sure if that's available freely. - For MVP, we could have a simple dictionary for a few test companies or just rely on a quick search result (like scraping Wikipedia or Yahoo). - *Bias handling:* The comparables approach itself has a bias in that market pricing of peers might also be wrong. DBOT (like Damodaran) uses it as a check but not the final say. So the system will not simply conform its valuation to the multiples of peers, but it will discuss them. If DBOT's DCF says $420 and all peers imply it should be say $300, DBOT will note that discrepancy and either justify it (maybe DBOT expects BYD to outperform peers) or acknowledge it as a reason to be cautious. This nuance should come out in the report (the Critic can ensure it's addressed).

In the BYD example: - BYD's relative valuation pointed to being cheaper than competitors [14], which supported DBOT's higher valuation (the market undervalued BYD relative to peers). - DBOT included a chart (Terminal EV/EBITDA comparison) [49] and emphasized this point. - Our system would replicate that: gather EV/EBITDA for BYD and peers, produce chart, mention it in report.

Combining with DBOT's value: The consensus and comparables primarily serve to **explain differences**: - If DBOT's value is above market and consensus target is also above market, that's a supporting factor (others also think it's undervalued). - If consensus is below market while DBOT is above, it means DBOT is contrarian; the report should acknowledge this difference and why DBOT's perspective might differ (maybe DBOT uses a longer horizon than analysts, or focuses on different metrics). - If peers are valued richly but our stock is not, DBOT highlights it as an opportunity (or consider if peers are overvalued outliers). - Conversely, if our stock is richly valued vs peers, DBOT might incorporate a more conservative stance unless justified by extraordinary prospects.

Integration in Orchestration: In the algorithm, both appear in the waterfall and possibly in the loop. After initial run: - The information gleaned (like "BYD is cheaper than peers" and "analysts predict lower growth") is fed into the router prompt. The LLM might decide to revisit one if something stands out (e.g., maybe consensus was drastically different, so do another sensitivity run to align with it). - If the router sees consensus and comparables are in line, it might focus elsewhere.

Outcome in Report: The final report will have sections or at least paragraphs on: - *"Analyst Consensus Check"*: where DBOT says e.g., "Analysts on average expect X, we assume Y; our valuation is higher because we anticipate [reason]. The consensus target price is $300, which is below our $420 estimate, indicating our more bullish outlook on BYD's growth prospects [109]." It will cite where appropriate (maybe cite Yahoo Finance for the consensus figures). - *"Relative Valuation"*: e.g., "Compared to peers, BYD trades at lower multiples (8.9x EV/EBITDA vs Tesla's ~17x) [14]. All else equal, this suggests BYD's stock has room to rise if it were valued on par with competitors. Our DCF value being higher than market aligns with this relative undervaluation." Possibly include a table or chart referenced as Figure (which Plotting agent created).

By addressing both consensus and comparables, DBOT's report covers both an **intrinsic perspective** (DCF) and **market perspectives** (what others think and how market is valuing similar assets). This triangulation is key for a credible analysis and also helps identify biases: - If DBOT and peers all disagree with market, maybe market is undervaluing the whole sector (which could be an opportunity or could reflect a risk factor like an entire sector out of favor). - If DBOT is the outlier against consensus and comparables, DBOT's user should question what DBOT assumes differently (and the Critic ensures the report explicitly points that out, preventing a blindspot).

In summary, the Consensus and Comparables agents incorporate **external reality checks** into DBOT's otherwise self-driven model: - They reduce the risk of DBOT straying into an unrealistic valuation by ensuring it's aware of external benchmarks. - They also provide rich content for the report's narrative, strengthening the rationale behind DBOT's conclusion or at least providing a balanced view ("others think differently, but here's why DBOT stands where it does").

8. Plotting and Report Specification

Plotting (Visuals): Visualizations make the valuation story easier to digest, and DBOT's Plotting agent is dedicated to producing these supporting graphics [52] [47]. Based on the analysis, we anticipate the following **key visuals**: - **Valuation Summary Chart/Table:** Possibly a waterfall chart or a table showing the different valuation outputs after each agent's contribution. For example, a table: - "Base DCF value: $450; After sensitivity alignment: $430; After consensus input: $425; After comparables: $420; Market price: $250." This was not explicitly shown in the paper, but could be helpful to show how we arrived at final value. Alternatively, a bar chart comparing DBOT's value to market price and consensus target. - **Sensitivity Analysis Results:** A chart or table showing how valuation changes under different scenarios. Since DBOT often returns a table of valuations for varying assumptions [32], we can plot that. For instance, a line or bar chart showing valuation under pessimistic vs optimistic assumptions (or a tornado diagram showing which driver has biggest impact). In BYD example, they mentioned a table range $417–$468 [87]; a simple table can be in the report text, but a small chart could illustrate DBOT's scenario range vs market price line. - **Comparables Multiples Chart:** As noted, a bar chart of BYD vs peers EV/EBITDA (or P/E). The paper explicitly references *"Figure 2: TEV/EBITDA Multiples of BYD and Competitors"* [49] in the BYD report. We should replicate such a figure: a bar for each company (BYD, Tesla, Nio, XPeng, Li Auto, etc.), maybe highlighting BYD's bar. This visual emphasizes BYD's relative valuation point. - **Historical Trends (if needed):** Possibly a graph of BYD's past revenues/earnings vs forecast to show continuity of assumptions. Or a stock price vs intrinsic value over time (if DBOT runs historically, could show how valuation changed). For MVP, maybe not needed, but something like deliveries over 3 years (the text mentions BYD's EV deliveries steady increase except a blip) [12]; if that data is easily available, a small chart could illustrate it. But likely we focus on forward-looking charts.

The Plotting agent receives data from I or from analysis outputs. We might design it such that: - The Sensitivity agent returns a JSON or table of values (like [scenario: growth+1%, value: $X] etc.). Plotting agent knows how to turn that into a chart. - The Comparables agent returns a list of {name, multiple} – easy to chart as bars. - Some charts might require a bit of calculation or formatting (like computing EV/EBITDA from I? But presumably comparables agent does it). - We can template these chart types in code. Possibly use Matplotlib: it's free and fine for static images. E.g., `plt.bar()` for comparables, `plt.plot()` for scenarios. - The agent could either have pre-defined plotting routines or dynamically use LLM to code them. The paper's approach of LLM-generated code [50] is very flexible, but possibly prone to errors. We might combine approaches: have a library of common plot prompts that the LLM can fill in details. Or simply implement directly for reliability in MVP (with maybe LLM adding labels or style). - For example, we might feed an LLM: *"Here is data: BYD 8.8, Tesla 17, Nio 15... Write Python matplotlib code to plot a labeled bar chart of EV/EBITDA for these companies, highlighting BYD."* Then run the code it outputs. We must sandbox and verify the code to avoid malicious or incorrect output, but since we control prompt, it should be safe. If it fails, fallback to a known good implementation.

Report Specification (Structure & Format): The final report should be professional and clear, akin to an equity analyst's report or Damodaran's blog-style valuation posts [112] [9]. We structure it as follows: - **Title & Heading:** A catchy title that encapsulates the investment thesis or context (the BYD example shows a pattern: *"Company in Year: [Positive angle] or [Negative angle]?"* to show balanced narrative [55]). The Writer agent can create this based on the content (as DBOT did multiple times with varying creative titles [113] [114]). Under the title, possibly a one-line tagline: e.g., "Riding the EV Wave or Struggling Through the Storm?" which hints at the conclusion (BYD has big growth but faces competition). - **Introduction:** A brief paragraph summarizing the company, current market price vs DBOT's valuation, and the stance (e.g., undervalued, implying a buy). This sets the stage. It may mention recent performance (stock up or down, key news). - **Background & Business Overview:** (Optional if needed, or integrated into introduction) – a bit about what the company does, its industry. DBOT might skip deep background if focusing on valuation specifics, but some context might be included for completeness. - **Valuation Summary:** Present DBOT's intrinsic value calculation with main assumptions. Could be a paragraph: "Based on a DCF analysis, we estimate BYD's intrinsic value at ~$420 per share, significantly above the current ~$250 [109] [115]. This assumes revenue growth of X%, improving margins to Y%, etc." Possibly include a table of key assumptions (if not too detailed for text). - **Drivers and Assumptions Discussion:** Possibly subsections for each major driver: - *Growth:* Discuss the growth outlook (supported by data, news: e.g., "Global EV market growth to 60M units by 2040 supports a long runway [110]"). - *Margins:* Discuss current/proj margins, any competitive pressure or efficiency gains. - *Reinvestment:* If notable (BYD had improving efficiency after heavy capex early, etc.), mention capacity expansion etc. - *Risk/Cost of Capital:* Explain WACC choice (market factors, beta, interest rates). - **Analyst Consensus Comparison:** A section noting what the market consensus is and how DBOT differs. e.g., "Analysts expect slower growth (8%) and have an average target of $300, reflecting more conservative assumptions. Our model's higher value stems from a more aggressive growth outlook, which we believe is justified by BYD's technological edge." – plus cite sources for those consensus numbers [109]. - **Comparables Analysis:** Present the relative valuation chart and insights [14]. E.g., "BYD's valuation looks attractive relative to peers: its EV/EBITDA is roughly half of Tesla's, despite comparable growth. Figure 2 illustrates this comparison, underscoring BYD's potential undervaluation [14]." - **Recent News & Impact:** Summarize crucial news items and how they affect the valuation. E.g., "Recent tariff concerns introduced uncertainty in the European market [108], which we accounted for by a slightly higher discount rate. Meanwhile, strong delivery numbers highlight BYD's solid execution [12], supporting our growth forecasts." - **Sensitivity Analysis:** Include the table or description of scenario range. "Our valuation remains robust under various scenarios: using more conservative assumptions (e.g., lower growth or margin), we get values in the high $300s, while optimistic scenarios yield ~$470 [11]. Even the low end exceeds the current market price, suggesting a margin of safety." - **Conclusion/Recommendation:** Wrap up with a clear statement: "DBOT's analysis indicates BYD is undervalued. With a stable intrinsic value ~68% above market [109] and consistent findings over multiple weeks [116] [117], this is a **Buy** recommendation for long-term investors." Possibly mention risks: "Key risks include political interventions and the pace of EV adoption, which could slow growth (as noted by Damodaran's critique) [118]." - **Citations and Sources:** Throughout the report, we include footnote-style citations (like Damodaran often references news or his own posts). Our system will use the format from our chatbot environment (like 【source†L#-L#】) to indicate sources of data or quotes included, as per user instruction. For example, citing a statistic from news or a line from the paper as supporting evidence. - **Length & Style:** The Critic will ensure the report is not overly verbose (one of Damodaran's feedback was to cut verbosity [119]). We aim for a concise but comprehensive report (perhaps 4-6 pages in text with charts, which is substantial but not excessive). The language should be formal but accessible, similar to a research report, and maintain neutrality/balance until the final recommendation (no hyperbole). - **Read-only data usage:** The Writer is constrained to read from I and analysis outputs; it won't introduce new data. The Critic checks that any claim corresponds to something in I/DC or news summary. This prevents hallucination and ensures traceability to our data.

Report Output Example (from BYD): The Appendix B excerpt suggests DBOT's report included: - The creative title "Riding the EV Wave or Struggling Through the Competitive Storm?" [55]. - Discussion of deliveries (which came from data/news) [110]. - A relative valuation statement: BYD's TEV/EBITDA 8.8x vs others higher [14]. - The four value drivers explicitly listed with their values [13] – likely as part of the text or a bullet list. - A sensitivity range noted [87]. - Ending note on strategy differences by region [120] – which is an insightful qualitative point that a good analyst would mention. - Also, Damodaran's critique highlights missing qualitative questions (EV adoption pace, government role, etc.) [121] [122]. We can preempt some of those in the report's risk discussion (like we did in conclusion mention EV adoption risk and government role briefly). Over time, the Critic agent can be tuned to ensure such qualitative factors aren't completely ignored by prompting the Writer to address them if obvious.

Citations: As per user's guidelines, every embedded chart from Plotting agent should be cited at the beginning of the relevant paragraph (with an `embed_image` reference), and the text should not explicitly mention the image source (the UI will show it). We will do something like: 【*chart_image†embed_image*】 *Figure 2: EV/EBITDA Multiples of BYD vs Competitors...* in the report text, and our system will have generated `chart_image`. In this environment, we can't actually generate an image here, but in the design doc we assume it.

"Read-only" Guarantee: The Report Writer agent having read-only access is a design guardrail [123]. That means when the LLM writes the report, it should not invent new numbers or change any values; it must pull from I (for numbers like 10% growth) and analysis notes (for statements like undervalued vs peers). To enforce this, the prompt can explicitly say *"Use only the provided data. Do not make up any financial metrics not given. If uncertain, refer to the Critic."* Also, the Critic will catch any inconsistency, e.g., if writer said margin 6% while I says 5.9%, the Critic will flag it for correction. This way, the final report is fully consistent with the model's data.

Layout: The output medium likely Markdown (since user asked for technical doc, but the actual system might output PDF or HTML). For design, we can assume markdown or a similar markup where headings, bullet lists, tables, and images can be included. The formatting guidelines from user indicate use of headings, lists, etc., which we will do in our output.

Tables: The report may include a table of the value drivers, or a small one for sensitivity. E.g.:

Assumption	**Value** (Years 1-5)	**Years 6-10**	**Terminal**
Sales Growth	10% (yr1, taper to 7%) [13]	7%	4.4%
Operating Margin	5% (next yr) [13]	6.7%	7.0%
Reinvestment (Sales/Cap)	1.2 (i.e., capex 83% of NOPAT) [124]	1.6	~1.6
Cost of Capital (WACC)	8.0% (assumed)	~8%	8%

Something like that could be in text or appendix if too detailed. The user only asked for JSON schema of I and v explicitly, which we provided earlier, but not necessarily for inclusion in the final report to an end-user.

Citations in Report: All factual claims or data points in the report should be cited to their source: - Financial data points → ideally EDGAR or Yahoo source. If our DC came from Yahoo, we might cite Yahoo Finance or a 10-K. Since we are writing a design doc, we use the sources we have (which in our case ironically is the DBOT paper text itself, but in actual system, it would be original sources). - News points → cite the news article or at least mention the publication/date, and in our environment we might use the news text as a source. - For design doc, we cite the DBOT paper where it shows those values as a proxy.

The Critic agent ensures all necessary citations are present, as per design [93].

In summary, the Plotting and Writer agents collaborate to produce a polished report: - **Visuals**: at least one chart for comparables, possibly one for sensitivity. These are referenced as figures in the text for clarity. - **Narrative**: logically structured sections covering all aspects of analysis, heavily supported by data and sources, and concluding with a clear actionable insight. - The style should match Damodaran's approach: thorough but not overly academic, with a mix of qualitative narrative ("stories") and quantitative analysis ("numbers") [30].

By carefully specifying format and using the Critic oversight, we ensure the output is not just comprehensive but also **readable and trustworthy** – hitting the user's goal of a high-quality valuation report.

9. Logging and Governance

As a financial AI system, DBOT must operate with transparency, accountability, and safeguards. We incorporate robust **logging, validation, and governance mechanisms** to monitor the system's behavior and ensure it adheres to guidelines.

Logging & Audit Trail: Every step in the multi-agent process is logged to an **Audit Log**: - **Data Retrieval Logs:** When data is fetched (DC built), log the source URLs, timestamps, and any data cleaning steps. For example: "2025-09-03 10:00: Fetch Yahoo Finance income statement for BYD – success" with perhaps a checksum of data. - **LLM Interaction Logs:** Each prompt sent to an LLM and its response are recorded (perhaps not in full in a final system due to cost, but at least a reference or key decision). The Router's decisions especially are logged: e.g., "Iteration 2: Router chose 'news' with instruction 'check new developments'." The actual prompt content can be stored in a secure log for debugging or audit if needed to see *why* an agent did what it did. - **Model Calculation Logs:** The output of f<sub>fcf</sub> (key interim results, final v) for each iteration is logged. For instance: "Base DCF v = $450; after consensus v = $430," etc. This helps in verifying the step-by-step adjustments. - **Report Generation Logs:** Note when the Writer agent was invoked and when the Critic finished. If any changes were made after Critic feedback, log those changes (like "Critic flagged missing source for growth 10%, added citation to 10-K"). - We will also keep a **manifest** of all files produced (report, images) for each run, with unique IDs or hashes, enabling later reference or comparison (e.g., comparing BYD report versions over time).

These logs enable **traceability**: one can reconstruct exactly how DBOT arrived at a conclusion, which is vital for trust. If a user or regulator asks "Why did DBOT recommend this?", we can show the chain of reasoning and data behind it.

Manifest & Versioning: Each run could produce a **manifest file** summarizing: - Company, date of analysis, version of DBOT code, version of model prompt set. - Data sources used and their versions (e.g., "Yahoo Finance data as of 2025-Q2, news up to 2025-09-03"). - Summary of final outputs (intrinsic value, market price, recommendation). - Any warnings or errors encountered. This manifest ensures any future auditing or backtest can align the results with the exact conditions under which they were produced.

Guardrails: We implement multiple **guardrails** to prevent undesirable behavior: - **Numerical Consistency Checks:** As mentioned, Critic or additional validators verify that important numerical outputs in the report match the calculations. For example, cross-check that the sum-of-parts (if any) equals total, or that if we claim a percentage difference, it's computed correctly (we might have small code functions to compute differences to avoid LLM doing it). - **Out-of-scope Prevention:** The system is instructed not to venture beyond valuation. If an LLM agent response starts veering (e.g., giving personal finance advice or discussing irrelevant topics), the Supervisor or Critic will catch that and truncate or refocus. Prompts explicitly define the task and forbid, say, any attempt to get non-public data or do anything unethical. - **Time limits:** Each agent has a timeout. If an agent fails to complete (LLM not responding, API hang), the Supervisor will log a timeout event and decide whether to retry or skip. This avoids a stuck system. - **Decision Bounds:** The Router LLM's domain is constrained by the options provided. It won't be allowed to execute arbitrary code or call unknown functions – the Supervisor only honors recognized agent names. If it outputs something nonsensical, we break out as per algorithm and proceed to finalize (with a possible note that analysis was truncated if that impacts results). - **AI Bias and Ethics:** DBOT's recommendations could influence investment decisions, so we must be careful. While not an immediate technical guardrail, governance includes ensuring DBOT does not systematically favor certain companies without basis. Because DBOT is trained on Damodaran, it inherently has a value investing bias [125] [126]. We acknowledge this bias in documentation and potentially measure it (the paper suggests analyzing how DBOT's biases load on factors). For governance, we could periodically analyze DBOT's outputs vs. a benchmark to ensure it's not drifting into a dangerous territory (like always recommending one sector, which might indicate an AI misinterpretation). - **Regulatory Compliance:** If this system were deployed widely, we'd need to consider regulations (e.g., is this investment advice? We might need disclaimers). For now, we ensure the report includes a disclaimer like "This is an AI-generated analysis for informational purposes." The Critic can append that if needed.

Security: All API keys (for data, for LLM) are stored securely, not exposed in logs. Data fetched and outputs are for public companies, so nothing confidential usually, but we treat it carefully regardless.

Audit Trail Example: For BYD report on 2024-11-04 (as in Appendix B): - The logs would show each week's analysis perhaps. They noted DBOT produced reports weekly with slightly different tone but stable valuation [111] [109]. We would have each run manifest, so one could see that on 2024-10-15, 10-22, 10-29, 11-04 what changed. The logs might show news changing (tariff news intensifying by early Nov, causing Critic to note more cautious language, etc.), while the valuation remained around $420 all times, which was logged, demonstrating stability. - If an error occurred (say on 10-22 news fetch failed), the log would note it and that DBOT proceeded with last news or none, so one could correlate any difference in output to that factor.

User and System Feedback: Governance also includes a feedback loop for improvement: - We allow human reviewers (or an automated monitor) to flag if any output seems off (e.g., a factual mistake, or an inappropriate statement). For example, if Critic somehow missed a factual error and it got to output, a human reading could flag it. Then developers can adjust prompts or add rules to catch that case next time. - Over time, DBOT's performance (if integrated in investment decisions) would be evaluated. If it consistently biases high or low or misses certain risk (like failing to consider management quality as the critique noted [127]), the governance process would mandate enhancements (maybe integrating a "Qualitative factor agent" in future).

Ethical and Legal Considerations: We should consider: - DBOT should avoid giving definitive "Buy/Sell" without context. It provides analysis. The current phrasing might lean to "buy recommendation" for clarity, but in deployment we'd include disclaimers as mentioned. - If DBOT text could be manipulated (prompt injection attacks in LLM, etc.), we guard by not letting external inputs go to LLM without sanitation. For news, we strip any malicious content (though unlikely in news; still, if an article had some weird prompt-like text, our summarizer should not execute it as instructions). - Ensure compliance with copyright/data usage: We use only free data; any content like news article text we use should be allowed (if not fully, we only use it internally to generate summary, and output is an abstract, which is typically fair use).

Transparency to Users: Possibly include in the report or as a separate output a summary of what data was used and if any issues occurred. E.g., an appendix: "Data sources: Yahoo Finance (financials), Financial Times (news on tariffs, Oct 20, 2024)". This fosters trust. Alternatively, the citations in text partly serve this.

Replacing Human Analysts? (A side note from the paper's perspective [128]): The governance should ensure DBOT is used as a tool, not blindly. Logging and oversight by humans can catch if DBOT ever does something like inadvertently use insider info (shouldn't happen with our data sources, but hypothetically if news had a leaked info, DBOT might use it not knowing it's illegal – governance might restrict certain kinds of info or at least flag unusual gains that could be suspect). These are broader considerations as the system grows.

In conclusion, logging and governance features are designed such that DBOT is **transparent, verifiable, and controllable**: - Every number and decision is documented. - Mistakes can be traced and corrected. - The system can be audited externally (by an internal compliance team or even regulators if needed) by examining logs and manifests. - Guardrails and the Critic ensure outputs meet quality and ethical standards, minimizing risks of misinformation or misuse.

10. Implementation Plan

We will implement DBOT in **Python**, leveraging its rich ecosystem of data and AI libraries, with an eye towards eventual cloud deployment.

Technology Stack: - **Programming Language:** Python 3.x (for its balance of simplicity, data handling, and ML/AI integration). - **Financial Data Access:** Use libraries like `yfinance` for Yahoo Finance data (free and easy for stock prices, fundamentals) [36]. For more detailed fundamentals, use **Alpha Vantage API** (with their Python SDK or HTTP requests) for income statements, etc. [37]. The SEC EDGAR data can be accessed via their **REST API** or a wrapper (like `sec-edgar-api`) for specific filings if needed. - **LLM API Clients:** We will use OpenAI's API (for GPT-4) via the `openai` Python library. Also consider Anthropic's Claude API (their Python client) if needed, possibly for the Writer agent as it might handle larger context or more nuanced writing. The design allows swapping these: we abstract f<sub>llm</sub> calls such that the model can be chosen per agent. For example: - Supervisor/Router, Valuation, and other reasoning tasks could use GPT-4 (for its logic strength). - Writer could try Claude (which sometimes yields more verbose output, but if better structured). - We will experiment in MVP which combination yields the best results, but ensure we can configure it easily. - **Prompt Templates:** Maintain prompts in a structured way, possibly external JSON or YAML, so they can be tuned without changing code. We'll version these prompt templates (like v1.0 initial set from the paper, future v1.1 refined from testing). - **Data Handling:** Use **Pandas** for manipulating financial data (e.g., computing historical growth, etc.), and possibly for assembling data for charts. - **Plotting:** Use **Matplotlib** or **Plotly** for generating charts as image files (Matplotlib for static ease). We may embed charts in the final Markdown/PDF. Also will evaluate using LLM for code generation for plots; in MVP we might manually code a few expected chart types for reliability, leaving LLM-generation as an experimental feature. - **Environment:** Initially run locally or on a single server/jupyter environment for development. For cloud, containerize using Docker so it can run on a VM or a container orchestration (Kubernetes) eventually. The modular design helps here. - **Cloud Considerations:** Eventually, to deploy at scale (valuing hundreds of companies daily), we will: - Use batch jobs or parallel processing (the heavy part is LLM calls, which are costly and not easily parallel if using rate-limited APIs, but we can distribute companies across time or multi-accounts). - Possibly integrate with cloud functions or a workflow engine for orchestrating multi-agent flows for each ticker. - Data caching in cloud database (like store DC in an S3 or database to not fetch repeatedly the same data). - Use a message queue or API endpoints if we turn DBOT into a service (e.g., send a request for a ticker, get back a report). - For now, MVP runs one company at a time in a script.

Model / Agent Matrix: We define which LLM model to use for each agent: - *Supervisor/Router:* GPT-4, Temperature ~0 (we want consistent decisions). - *Valuation & Sensitivity reasoning:* GPT-4 or GPT-3.5 if cost is an issue, but GPT-4 likely needed for complex reasoning over spreadsheet context. - *News summarization:* GPT-4 can handle multiple articles summarization well, but GPT-3.5 might suffice for headline filtering. Possibly do a mix: use cheap model for initial headline filtering, and GPT-4 for final summary. - *Plot code generation (if used):* a GPT-4 with code knowledge (the OpenAI code model or GPT-4 itself is quite good at code). - *Writer:* Possibly Claude (Anthropic's model) if it produces well-structured text. The paper suggests Claude might be better for writing tasks [76]. If using Claude, ensure the context (I and data) can be provided fully. If GPT-4 is doing fine, we might stick to one to minimize complexity. - *Critic:* GPT-4 ideally, since it needs to understand the whole report and catch subtle errors. Could also attempt GPT-3.5 first for cost, but likely GPT-4 for best results.

We'll monitor token usage and cost. DBOT's process can be token-heavy (with financials, tables, etc.). Hosted LLMs have context limits (GPT-4 up to 8K or 32K tokens). If our I is large (say 10K tokens if we included full financial statements), we might condense data for the LLM. Training a custom model is not in scope (and likely unnecessary given available models), but if needed for scale, one might consider fine-tuning or using open-source local models eventually. Initially, the hosted APIs suffice.

Development Steps (MVP vs v1.1): - **MVP (Minimum Viable Product):** 1. Implement the DCF calculation function f<sub>fcf</sub> and test it standalone with sample inputs. 2. Implement data fetch for a small subset (e.g., using yfinance to get last FY revenue, earnings, etc., and maybe manually input few assumptions to feed f<sub>fcf</sub>). 3. Integrate one LLM agent at a time: e.g., start with just Supervisor calling Valuation (which might not use LLM at first, just do f<sub>fcf</sub>). Verify we can produce a basic report with just that (like a simple output: "Value = X, vs price Y"). 4. Add Consensus and Comparables: fetch some dummy consensus (or skip if not easily available in free data at first) and peer multiples (maybe manually define 2-3 peers for test). 5. Add News: perhaps use a dummy news summary or a single known article to test the pipeline. 6. Add Writer and Critic to assemble it nicely.

In MVP, we may hardcode some parts to ensure end-to-end works, then replace with dynamic. For example, for testing BYD, we might pre-fill DC from known data rather than rely on all APIs, just to test integration.

The goal of MVP is to generate a BYD-like example report successfully, even if some data was stubbed. - **v1.0 (Initial Complete Version):** After MVP, flesh out all integrations fully: - Full data retrieval automation (actual API calls for consensus if possible). - Robust news fetching code. - Fine-tune prompts and get all agents functioning as intended with real data for a couple of companies (maybe test on a US company too, like Apple or something, to ensure generality). - Ensure logs, error handling, etc., are implemented. - Optimize runtime: maybe parallelize the data fetch (financials and news can fetch in parallel while LLM doing initial mapping? Not critical but could consider). - **v1.1 (Extensibility and Improvements):** Based on feedback and initial usage: - Expand data sources: e.g., incorporate a macro data fetch (like automatically get the latest 10-year treasury for risk-free). - Caching layer: implement a small caching mechanism for API results (so if we run S&P500, we don't hit Yahoo 500 times for the same data daily). - Concurrency: ability to run multiple valuations concurrently (maybe multi-thread or multi-process, mindful that LLM API calls are network-bound so can do async). - UI/Delivery: Possibly create a simple web interface or scheduled report generation. For cloud deployment, wrap the code in a Flask API or CLI so that it can be triggered easily. - Incorporate Damodaran's historical data if possible (like perhaps training an internal model to help f<sub>vd</sub> or identify framing questions from his past valuations – but that's researchy; likely not in 1.1, maybe later). - More rigorous testing with backtesting: e.g., run DBOT on several past dates (with historical financials) and see how its picks performed. That's more for validation research rather than implementation, but it's a possible path.

Testing: Develop **acceptance tests** (like the BYD test in Section 11) and unit tests for components: - Unit test f<sub>fcf</sub> with known inputs and outputs. - Test each agent function with controlled inputs (maybe simulate an LLM by using a fixed prompt response for determinism). - Integration test: run the whole pipeline on a small company with easily verifiable data (maybe a stable company where we can manually compute a quick DCF to see if result similar).

Model Monitoring & Updates: Keep track of LLM API changes or costs. If an API becomes too expensive or rate-limited, consider alternatives or optimizing calls (maybe reuse context across agents if possible to reduce tokens, though each agent has distinct role so maybe not trivial). For long-term, consider an on-premise LLM if it gets good enough to avoid API usage – but currently GPT-4/Claude quality is high and difficult to match open-source.

Documentation & Version Control: We will maintain documentation for how each agent is implemented and how to configure prompts or run the system. Use a version control system (git) to track changes in code and prompt configurations. Tag versions (v1.0, v1.1, etc.) for reference. Possibly create a repository for "dbot" where all this lives.

Deployment Plan: - After testing locally, we can deploy to a cloud (e.g., AWS or Azure): - Use an EC2 or Azure VM with Docker container running DBOT to generate reports on demand. - Or use an Azure Function/ AWS Lambda for each analysis, though the process might be too heavy for serverless due to LLM call latency (likely a persistent worker is better). - For scaling to many companies, possibly schedule tasks in a queue and have a pool of workers (since each analysis can take maybe a couple of minutes because of multiple LLM calls). - Ensure secrets (API keys) are stored securely (e.g. in environment variables or vault). - Logging in cloud: push logs to a storage or logging service (CloudWatch, etc.) for analysis.

Summary: The implementation plan leverages proven tools (Python libraries, GPT APIs) to build DBOT step by step, starting with a single example and extending to a robust system. The modular agent architecture aligns well with Python's flexibility, and with careful prompt and API use, we aim to have a functional prototype soon that can be iterated on. The plan emphasizes maintainability (clear separation of concerns per agent, externalized prompts) and scalability (cloud-ready design, caching and concurrency for bigger workloads).

11. Acceptance Tests

To validate DBOT's functionality and performance, we design a series of acceptance tests. These tests will ensure that the system meets the requirements using a concrete example (BYD) and other cases, and that each component behaves correctly under expected and edge conditions.

Test Scenario: BYD (November 4, 2024) – This is the primary end-to-end test as it replicates the Appendix B example: 1. **Input Setup:** Prepare or fetch BYD's data as of 11/4/2024. This includes: - Financial statements up to 2024 (e.g., FY2023 results if available, or trailing 12 months by Q3 2024). - Market data: stock price around that date (~HK$250, which matched the narrative [55]). - Consensus: say analysts 5-yr growth ~8%, target price HK$300 (from sources around then). - Comparables: Tesla's multiples around that time (let's assume EV/EBITDA ~17x), and Chinese peers Nio, XPeng ~12-15x. - News: Key news up to early Nov 2024 – e.g., EU tariff announcements in Oct 2024, BYD Q3 sales numbers, US election looming with tariff talk. These data can be stubbed or loaded from a snapshot. 2. **Execution:** Run DBOT on BYD with that data (through the normal pipeline). 3. **Expected Outcome:** Check that: - The final **intrinsic value** output (v) is in the expected range ($417–$468 range, specifically around $420) [87] . - The final report contains the crucial elements: - Title similar in style (the exact wording may differ but should be similarly insightful). - Discussion of deliveries and growth (e.g., mentions the steady increase to 8M EV and projection to 60M by 2040) [110] . - Relative valuation stating BYD's TEV/EBITDA ~8.8 vs peers higher [14] . - The four driver assumptions listed (growth, margin, etc. matching values given) [13] . - Sensitivity analysis range included (417 to 468) [87] . - Conclusion that it's undervalued and likely a buy (with maybe wording about recommendation). - Verify all numbers in the report match the data (for instance, if I had growth 10%, the report says 10%, not some other number). - Check citations: each major factual claim is cited to a source (our test data or reference texts). - **Pass/Fail Criteria:** If the report's content matches the above expectations and v ≈ 420, the test passes. Deviations like a vastly different v (say 300 or 600) would be a fail indicating something went wrong in the model or inputs. Missing key discussion points (like not mentioning comparables or not addressing why DBOT's value is different) would also be a fail as it means the agents (Writer or Critic) missed required output.

This BYD test essentially verifies the entire pipeline in a realistic scenario. We will also include some **quantitative assertions**: - The valuation engine should produce about 420 given those inputs: we can compute separately outside LLMs a simplified DCF to see if it aligns. - The stable iteration: ensure that if we run DBOT for BYD multiple times with same inputs, the output is the same (or differences only in wording, not in conclusions or numbers). This tests determinism/stability.

Test: Different Prompt Styles (Robustness): Although the user likely always uses the same invocation, test that slight changes (like asking for a different output format) do not change the core analysis. E.g., if we asked for "summaries only", would it still compute v similarly? This is more of a system robustness test rather than typical usage. But since our input is fairly structured (ticker, date), not much user prompt influence exists in our design, which is good.

Edge Case Tests: - **No News Scenario:** If a company has no significant recent news (or offline mode), test that the News agent gracefully outputs "No major news" and the rest still runs. For example, pick a stable company with no news and run DBOT. Pass if it doesn't crash and just omits the news section politely. - **Data Missing:** Simulate missing data in DC (like no consensus info available). The Consensus agent should not crash; it might output "Consensus not available" or simply skip adjustments. The report might note "Analyst consensus data was not found." We expect the system to still produce a report, maybe with a caveat. Pass if so; fail if it errors out or leaves blanks. - **Negative Earnings Company:** Test on a company with negative earnings (to see how model handles possibly negative FCF in initial years). For instance, a startup or a biotech. DBOT should still produce a valuation (likely focusing on future turning profit). Check that the Valuation agent or f<sub>fcf</sub> doesn't break (like terminal value can still be computed if eventually positive flows). - **Financial Company (Out-of-scope):** If we unintentionally run on a bank or insurance company, see what happens. Likely our model isn't tuned for that (DCF for banks is different). We'd expect either DBOT to proceed with a flawed model or possibly the Supervisor could detect and warn "Valuation for banks not supported." Since we haven't implemented such detection, it might produce output that is not meaningful. This is an identified non-goal, but we should test to ensure it fails gracefully rather than silently giving wrong info. A pass might be to at least get a report that highlights uncertainty or clearly states assumptions that are shaky (the Critic might say "This is a financial institution, results may be unreliable").

Performance Tests: - **Back-to-Back Runs:** Run DBOT on multiple tickers sequentially (maybe 5 in a row). Ensure caching works (if implemented) so that if two companies share a peer or something the second run might reuse data, etc. Also ensure no state bleed: each run's data should not leak into the next (the Supervisor should reinitialize I properly each time). We check log or outputs for any cross-talk. Pass if each is correct and independent. - **Time to Completion:** The process should complete within a few minutes for one company. We set an acceptance threshold, say <= 5 minutes on average. For BYD test, measure from start to final report. If it takes, for instance, 15 minutes, that might be too slow (fail, needing optimization or reducing LLM calls). - **Convergence Behavior:** For a test company, deliberately set high differences to see if the loop stops appropriately. Perhaps modify I initial vs market to see multiple loop iterations. Ensure it doesn't loop more than, say, 5 times and stops. If we detect an infinite loop scenario (like toggling), we'd fail that test and refine the router prompt or break logic.

Quality Gates (Pass/Fail): - Each test above should have clear criteria: - *Content correctness:* The final output must contain specific expected info (like BYD's known values). - *Stability:* Repeated runs yield same core results. - *Graceful degradation:* Missing data doesn't cause crash. - *No hallucination:* In any of these tests, if the system outputs a fact not supported by DC or news (e.g., makes up a financial metric), that's a fail. We rely on Critic to catch these. - *Citations:* If any statement (like "deliveries increased except a blip due to tariffs") is made without a source when we had one, that's a fail of the Critic or Writer (lack of citation). - *Report readability:* Though subjective, we want to ensure the report is logically organized. Perhaps have a human evaluator or a checklist to verify the sections are present. We can also use a simple heuristic: headings present, no obviously unresolved placeholders (like the LLM didn't leave "<figure here>" or something). - *Formatting:* Verify that embedded images appear and are referenced properly in text.

We'll document the results of acceptance tests and fix any issues found: For example, if BYD test shows the writer included too much verbose text, we adjust prompts to be more concise (which addresses Damodaran's note on verbosity) [119]. If consensus wasn't integrated, maybe the prompt needed tweaking.

Automated vs Manual Testing: Initially, these acceptance tests will be somewhat manually examined (reading the report). Over time, we can automate parts: - After generating BYD report, have an automated script search for certain keywords or values (like "8.8x" or "60 million by 2040"). Or parse the Markdown output to verify presence of expected sections. - For numeric outputs, check that final intrinsic value is within a reasonable bound of some benchmark calculation.

Regression Testing: Keep the BYD scenario and possibly others (Apple, Tesla, etc.) as regression tests. Whenever we update code or prompts (v1.1, v1.2...), re-run these tests to ensure outputs didn't degrade (unless intentionally changed). Logging the outputs or summary metrics allows comparison (e.g., if previously BYD value was 420 and now after changes it's 500 with same data, investigate why – maybe a bug introduced).

By designing thorough acceptance tests around a high-profile example and edge conditions, we ensure DBOT is reliable and ready for broader use. The BYD case in particular gives confidence that the system can replicate what was documented in the research paper, which is a strong validation of correctness.

12. Risks and Mitigations

Implementing DBOT entails several **risks** – technical, operational, and ethical – which we identify and address with mitigation strategies:

1. LLM Hallucination or Error: The LLM agents might produce plausible-sounding but incorrect statements (e.g., misreading a table, or hallucinating a news fact). - *Mitigation:* The **Critic agent** double-checks all factual content [59] . Also, wherever possible, we rely on deterministic computations (the Python valuation engine) for numbers, and have the Writer insert those numbers directly from I rather than asking the LLM to calculate. We instruct prompts to be truthful and if unsure, to refer to data or say nothing beyond the data. Additionally, by providing the LLM with the actual data context (e.g., include relevant parts of financial statements or news text in the prompt), we reduce chances of it making something up. If hallucinations are detected in testing, we refine prompts or break tasks into smaller pieces to isolate and verify output.

2. Data Quality and Availability: The system depends on free data sources that may have limitations. Data could be outdated, incomplete, or an API could go down. For example, Yahoo might occasionally provide stale data, or consensus info might not be present. - *Mitigation:* Use multiple sources where feasible (cross-verify key data like revenue or price from two sources). Implement checks in DC building: if one API fails, try an alternative or use cached last known values. For consensus, if not available freely, have the system note that and possibly proceed without it, rather than importing potentially wrong data from an unreliable source. Logging any missing data scenario will help us later fill those gaps or adjust the approach. For free API limits (Alpha Vantage's 5 calls/minute limit [129]), implement caching and maybe a queue to throttle requests. If scaling up usage, consider obtaining higher-rate API keys or using an alternate like scraping a site with proper caching.

3. Overfitting to Damodaran Bias: DBOT is designed around Damodaran's methodology, which has a known bias towards value investing and possibly missing qualitative factors (as noted, e.g., ignoring momentum, or underestimating paradigm shifts) [127] [130] . This could mean DBOT systematically undervalues high-growth tech with no profits or sells winners too early (like Damodaran did with Nvidia) [125] . - *Mitigation:* Acknowledge this bias in documentation (so users know it's by design somewhat). We can incorporate certain **counter-bias checks**: for instance, an agent could be added in future that considers momentum or qualitative factors such as brand strength (if news mentions it, etc.). In the risk section of each report, we ensure to mention factors DBOT might not fully capture (e.g., "Our model is fundamentals-driven and may not account for investor sentiment that can keep prices high"). For internal testing, we could compare DBOT's output to actual outcomes over time to see if bias leads to underperformance, and then adjust methodology (maybe tweak cost of capital or growth estimation to align better with reality, effectively "learning" from performance). In governance, having a human expert occasionally review outputs can catch cases where a purely quantitative model misses the story – those can guide prompt adjustments.

4. Underdetermination & Multiple Valid Narratives: The paper demonstrated DBOT could produce both bullish and bearish reports for BYD with the same numbers [131] [132]. This means the LLM might be capable of spinning the narrative in different ways, which poses a risk: lack of consistency or injecting an unintended bias if prompted differently. - *Mitigation:* Control the **prompting and temperature** to maintain a consistent narrative approach. The instructions to the Writer agent will emphasize an objective tone: incorporate both positives and negatives, but ultimately stick to the data-driven conclusion. By fixing the prompt and using low temperature, we reduce variability – the example with bullish/bearish was likely forced by prompt engineering to test it. We won't ask DBOT to "argue bearish" unless needed; in normal use it should just follow the logical path (which typically will lean one way if data suggests undervaluation). The Critic also ensures no "loose ends" [59] – which includes making sure the narrative isn't internally contradictory or flip-flopping in tone.

5. Systemic Risk / Misuse: If such AI systems were widely used, one could imagine scenarios where many agents trade on similar signals, potentially amplifying market moves (flash crashes, etc.) [133] [134]. While initially DBOT is a research prototype, it could be misused by users who blindly follow it or integrate it into auto-trading. - *Mitigation:* Build **safeguards and disclaimers**: e.g., "DBOT is not a licensed financial advisor. Use outputs at your own risk." At a technical level, we are not connecting DBOT to execution – it only generates analysis. In a deployment, one might impose rate limits (not valuing thousands of stocks instantly to move markets) and monitor if outputs are being used for trading in a way that could cause market impact. This is more of a policy/regulatory area – for our design doc, we note the importance of potential future regulation and plan to cooperate (perhaps log DBOT's recommendations separately if needed for compliance).

6. Regulatory/Compliance Issues: Generating investment reports could have legal implications (e.g., considered investment advice, needing disclosures). Also, DBOT using data (like news or financial statements) must respect copyrights and terms. - *Mitigation:* Ensure that only publicly available information is used (which we do). Include necessary disclaimers in reports (the Critic can add: "This report is AI-generated and for informational purposes. Not investment advice."). If deployed commercially, one would likely have a compliance review of the output format and content (like a person checking it doesn't violate any regulations about stock promotion or something). We should also be careful with material non-public info – since we only use public, this is fine; just ensure we don't inadvertently incorporate something like an insider comment from a forum (we won't, since we stick to reputable news). - Also, the reports will need to avoid any defamatory or overly speculative language. The Critic can ensure language is measured and data-supported, reducing liability issues.

7. Technical Failures & Recovery: The orchestration could fail mid-run (e.g., network glitch connecting to LLM API). - *Mitigation:* Implement retries for transient failures. Use try/except around API calls and have fallback behaviors. If an LLM call fails, perhaps try once more; if it still fails, log it and skip that agent or use last known state. The system should fail gracefully by still producing a partial report rather than none. E.g., if news agent fails, produce report without news but mention "we encountered difficulty retrieving news at this time." - On a larger scale, if deployed, have monitoring: if DBOT crashes or stalls, auto-restart or alert an engineer.

8. Cost Management: LLM calls (especially GPT-4) are expensive, and data APIs might also cost if scaling beyond free tier. - *Mitigation:* Optimize prompt sizes (don't feed unnecessary text), use cheaper models where acceptable (maybe GPT-3.5 for some tasks), and cache results. Monitor usage: if cost per company is too high, consider alternatives (like partial fine-tuning or using embeddings to let LLM search its own knowledge – e.g., FinGPT style – perhaps not in current scope but an idea for future to reduce API calls). For now, clearly define a budget and test how many companies can be done within it, and throttle accordingly.

9. User Trust and Understanding: Even if DBOT is accurate, users might not trust an AI's numbers or might misinterpret its analysis (especially given it can be complex). If a user doesn't understand how DBOT arrived at X, they might distrust or misuse it. - *Mitigation:* The transparency (citations, breakdown of drivers, etc.) is key to building trust [102] [103]. We also consider adding an **explainability module**: e.g., a simplified summary or Q&A interface where a user can ask "Why is the valuation so high?" and DBOT can answer referencing parts of the report (this is beyond initial scope but a direction). At least, in the report, we explicitly list the drivers and their values [13] so readers see what assumptions drive the outcome, which aids understanding and trust. - In governance, we might also simulate adversarial usage – e.g., what if someone tried to prompt-inject by renaming a company to include a trick phrase? Our system is mostly autonomous so not prompt injection susceptible via user input, but we consider any such potential and keep the LLM focus constrained.

By identifying these risks early, our design incorporates multiple layers of defense: rigorous checking (Critic, logging), constraints (prompt design, allowed actions), and transparency (citations, logging) to mitigate them.

No complex AI system is without risk, but with these measures, we strive to make DBOT's recommendations **reliable, reproducible, and well-bounded**, such that users can benefit from its insights with confidence in its underlying process [102] [135]. The continuous feedback loop via acceptance tests and human oversight will further ensure that as DBOT evolves, it does so in a safe and controlled manner, closing any gaps that come to light.

---

[1] [8] [17] [18] [19] [20] [21] [22] [23] [24] [25] [28] [29] [30] [31] [32] [33] [34] [35] [38] [39] [40] [41] [42] [44] [45] [46] [47] [48] [49] [50] [51] [52] [53] [54] [58] [59] [60] [61] [62] [63] [64] [65] [66] [67] [68] [69] [70] [71] [72] [73] [74] [75] [77] [78] [79] [80] [81] [82] [84] [85] [86] [88] [89] [90] [91] [92] [93] [94] [95] [96] [97] [98] [99] [100] [104] [123] [131] [132] [133] [134] dbot-paper.pdf
file://file-8sa5VzJM3kiTFDCxZV49Wj

[2] [3] [4] [5] [6] [7] [101] [102] [103] [126] [127] [130] [135] Papers by Jo~ao Sedoc
https://www.aimodels.fyi/author-profile/joao-sedoc-b17e4b3e-0b61-4dfe-9da3-5db27cfa69df

[9] [10] [11] [12] [13] [14] [15] [16] [26] [27] [43] [55] [56] [57] [76] [87] [105] [106] [107] [108] [109] [110] [111] [112] [113] [114] [115] [116] [117] [118] [119] [120] [121] [122] [124] [125] [128] [2504.05639] DBOT: Artificial Intelligence for Systematic Long-Term Investing
https://ar5iv.labs.arxiv.org/html/2504.05639v1

[36] Get Financial Data from Yahoo Finance with Python - GeeksforGeeks
https://www.geeksforgeeks.org/python/get-financial-data-from-yahoo-finance-with-python/

[37] [83] [129] Fundamental Stock Data in Python | Medium | Medium
https://hhundman.medium.com/fundamental-stock-data-in-python-5e04927e40c4