

Local LLM-Based Agentic System for Social Media Content Generation and Analytics

Aditya Kumar¹, C G Darshan², Chinmai Goutham K V³, Md. Arbaz Alam⁴, Suresh Kumar C⁵

¹²³⁴Don Bosco Institute of Technology, Visvesvaraya Technological University, Bengaluru, India

⁵ Assistant Professor, Computer Science and Engineering, Don Bosco Institute of Technology, Bengaluru, India

Abstract: Social media marketing requires timely, consistent content that is aware of the platform and this is labour intensive and hard to automate without sacrificing authenticity. In this paper we present an agentic web service that automates the end-to-end pipeline of social media content generation, analytics and insight reporting for small and medium enterprises, startups, creators and agencies. The system is a LangGraph state-machine orchestrator written in TypeScript, with supporting Express REST API and React dashboard. All content generation is powered by Qwen3-4B, a large language model with open weights, which runs locally on an OpenAI-compatible llamafile endpoint (i.e., no cloud inference). For LinkedIn, Twitter, Instagram and Facebook we distill different tone, length, structure, hashtag, call-to-action and audience constraints into a platform-strategy service and gather them in system prompts. Supervised curation graph process, validates and clusters content from Twitter, Reddit, GitHub and general sources. Analytics are computed from CSV datasets normalized into a SQLite schema and provide reach, engagement and sentiment metrics through REST endpoints and visual dashboards. A local Qwen instance is fed a structured summary of computed metrics and outputs findings, issues, and recommendations. It supports two modes of authentication to automate posting to Twitter and LinkedIn.

Keywords: Social Media Automation Content Generation Analytics Large Language Model LangGraph Agentic System.

I. INTRODUCTION

Organisations are turning to social media more and more to build brand visibility, engage audiences and acquire customers. However, maintaining a steady, platform-suitable posting rhythm takes a lot of manual work. Every network has its own standards regarding tone, length, structure, and hashtag and call-to-action style. Small and medium enterprises (SMEs), startups, creators and agencies don't always have the dedicated staff to produce and schedule tailored content at scale, while generic content-management tools are not able to adapt output to the specifics of each channel. Recent advances in open-weights large language models (LLMs) have enabled local, private content generation without the recurring cost, latency, and data-governance issues of proprietary cloud inference. You can keep control of the content in-house, you don't have to rely on the network

for generation, and you aren't charged per token if you run a decently-sized model on a consumer workstation. This manuscript describes a Social Media AI Agent, a full-stack agentic system that automates content creation, analytics, and insight reporting around such a local model.

The system models its workflow using the LangGraph framework, which models a generation or analysis pipeline as an explicit graph of typed nodes and conditional edges. This declarative formulation makes the control flow inspectable, testable and reusable, and enables multiple pipelines, the monolithic Main Agent and a parallel Supervisor to co-exist and share subgraphs. The rest of the manuscript is organised as follows. Section II describes the architecture, generation pipeline, local inference integration, content curation, analytics computation, insight synthesis and platform clients. The results of the exercise of the system are given in Section III, the design trade-offs and limitations are discussed in Section IV and Section V concludes the work.

A. Problem Statement

Traditionally, the creation of platform-specific social media content, calculation of marketing analytics from raw data and translation of those analytics into actionable recommendations are separate, manual tasks. The existing automated solutions are generally based on a single hosted language model, available via a closed API, which binds the user to external inference providers and inflexible, non-configurable templates. There is a need for a unified platform that (i) produces native and platform-aware content from a local model, (ii) consumes and normalises heterogeneous analytics datasets, (iii) calculates marketing key performance indicators, and (iv) interprets these indicators to generate data-grounded insights.

B. Objectives

The main objectives of the project are:

- 1) Build a local LLM content generation service without cloud inference;
- 2) Incorporate content strategies for each platform and business type into prompts;
- 3) Support CSV data ingestion with auto column normalisation in SQLite;
- 4) Do overview, platform, trend, content and sentiment analytics through a REST API;
- 5) Create a responsive dashboard to visualise these metrics;
- 6) Generate data-driven insights with the local model;
- 7) Allow authenticated posting to Twitter and LinkedIn.

C. Scope and Contribution

The original contributions in this study are in one cohesive, local-first agentic design which combines all aspects of graph-based orchestration (orchestrating), prompt-based platform strategies, analytics processing and insights production into one singular deployable web service. In addition to demonstrating how an open weights language model with approximately 4 billion parameters can generate both creative output and provide interpretive analysis for that same output, the project also demonstrated how multiple sources of curated content could be used in parallel to supervise the creation of a composite content item utilizing LangGraph send primitives.

II.RELATED WORK

Recent progress in transformer-based language models has achieved effective natural-language understanding and content creation. The Transformer architecture [1] on which modern large language models are based, and BERT [2] with its powerful bidirectional language understanding. GPT-3 [3] showed that large language models can perform a variety of generation tasks with few-shot learning.

LoRA [4] and QLoRA [5] have been proposed as efficient methods for adapting LLMs by reducing the computational and memory requirements for fine-tuning. Methods such as InstructGPT [6], Self-Instruct [7] and DPO [8] are instruction-following, and help to make a model more controllable and aligned. Open-weight models such as Qwen2 [9] and Qwen2.5 [10] are suitable bases for local AI applications. Llama 2 [11] and Zephyr [12] further show the potential of open and instruction-tuned models.

Batrinca and Treleaven [13] surveyed social-media analytics techniques for social-media applications. Hayat et al. [14] reviewed deep-learning approaches for social-media analysis. However, existing works mainly focus on language generation, model adaptation, or analytics separately. Tigma fills this gap by combining content generation (local LLM-based, LoRA/QLoRA adaptation, image generation), sentiment analysis, social-media performance analytics and publishing in one platform.

Work	Contribution	Tigma Gap
Transformer [1]	Attention architecture	No social-media integration
GPT-3 [3]	Content generation	No local analytics
LoRA/QLoRA [4][5]	Efficient adaptation	No complete platform

Work	Contribution	Tigma Gap
Qwen [9][10]	Local language models	No social workflow
Analytics [13][14]	Sentiment and performance	No content generation
Tigma	Generation + analytics	Unified platform AI

III. METHODOLOGY

A. System Architecture

The application includes all the files (backend, frontend, etc.) in one repository. This repository contains a typescript backend with express application that has routes for /health, /analytics, /generation, /dataset and /insight. In addition it will serve static files from its own static folder. At start-up time, the app creates the database schema into sqlite. Finally, it delegates generation and insight tasks to a Qwen3-4B model running on the same machine via an openai compatible llamafile service over localhost.

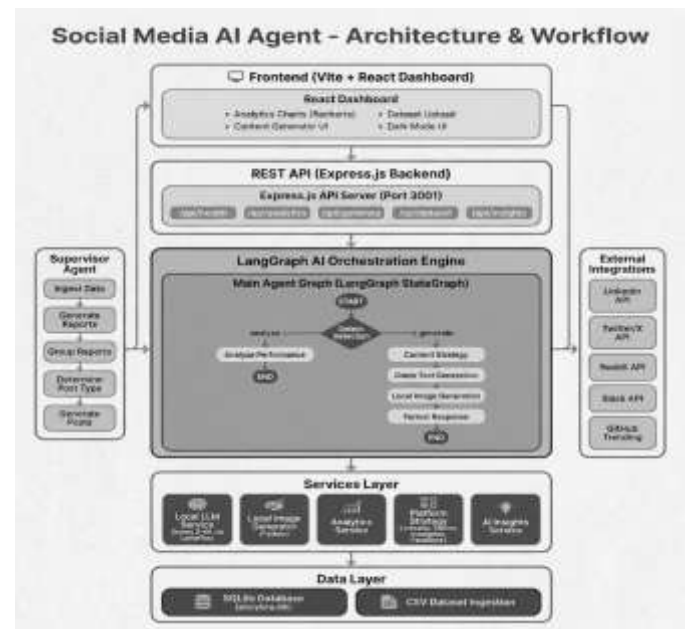


Fig. 1. High-level system architecture.

The frontend is an SPA (Single-Page Application) built with Vite and React. The Backend will run on a configurable port (default port=3001), but it's running a server for the Frontend using the Vite Development Server. For local development, cross-origin requests are allowed and the backend accepts JSON request body payloads of max size 50MB which can allow larger datasets to be uploaded. The environment variables that provide the Model Endpoint, the Image Generation Path and Provider Credentials are loaded from a .env file via

DotEnv at start-up. The LlamaFile endpoint is listening on the LoopBack Interface. A High-Level Architecture diagram is provided in Fig 1

B. Graph-Based Orchestration

The Main Agent is a LangGraph StateGraph that drives content creation by representing the pipeline as a series of typed nodes and conditional edges over an explicit state schema. The state schema defines fields for user request, business type, target platform, detected intent, content strategy, generated text, image prompt and paths, validation results, analytics data, final response and error information.

The workflow starts with an intent-detection node which classifies the user request using the local Qwen model. The prompt asks the model to return a strict JSON object with an intent of either `generate_content` or `analyze_performance`, a business type, a target platform and an image-inclusion flag. The raw response is stripped of any markdown fences before being parsed as JSON, with a fallback classification in case it fails to parse. A conditional router then routes analysis requests to an analysis node and generation requests through the strategy, text generation, image generation and response-formatting path. The formatted response is returned as a structured object with the platform, generated text and optional image URL as expressed in (1).

$$R = P+T+I$$

where P = platform, T = text, and I = imageUrl.

A parallel Supervisor graph extends the system to multi-source curation. First it calls the curate-data subgraph, then fan-outs report-generation tasks across content groups using LangGraph Send primitives, groups the reports produced, determines the post type and finally generates the posts. The fan-out allows parallelizing report creation on tweets, GitHub trending repositories, general web content and Reddit threads, each of them generating a separate report which will eventually be merged into the final assembled output

C. Local Language Model Integration

The local inference service represents the chat-completion endpoint of a Qwen3-4B-Q4_K_M model served by llamafile, compatible with the one provided by OpenAI. Configuration is driven by environment variables, with a default of the loopback address on port 8080. The service exposes three operations: a health check which probes the models listing endpoint with a short timeout; a chat-completion method which posts a system-and-user message sequence with configurable temperature and maximum token count and a two-minute timeout to accommodate local inference; and a single-question helper which prepends an optional system prompt before invoking the completion service. Failures return structured error objects that can be

surfaced to the user by the caller, rather than allowing unhandled exceptions to propagate.

1) Intent Detection Implementation

The intent-detection node sends the raw user request and a system prompt that includes the allowed business types (sme, startup, business, creator, agency, nonprofit), platforms (linkedin, twitter, instagram, facebook) and intent labels to the local model. The JSON response from the model is parsed to set the detected intent, business type, target platform and image-inclusion flag. If the request refers to an image, picture or photograph the include-image flag is set and propagated to later nodes. On any parsing failure the node defaults to sensible defaults, setting intent as generation with the default SME business type and LinkedIn platform.

2) Strategy Construction

The strategy node fetches the platform-specific system prompt for the selected platform and business type. It uses the platform-strategy service that considers tone, length, structure, hashtag, call-to-action and audience constraints and generates a prompt that instructs the model to produce only the post content, follow the maximum character count, and avoid meta-commentary such as opening phrases. The generated prompt is then used as the system message of the text-generation node.

3) Text Generation

The text-generation node calls the local completion service, passing the compiled strategy as system message and the user request as user message, with an elevated sampling temperature to encourage variation among regenerations. On success the generated text is trimmed and stored in the state, on failure an error field is populated and propagated to the HTTP response layer.

D. Platform Strategy and Generation

Platform strategy is formalized in a separate service that stores a profile for each target network with tone, maximum length, structural template, hashtag count, call-to-action style, media format and audience hint. Business-type modifiers help to further refine the voice for SME, startup, enterprise, creator, agency and non-profit contexts. Table I shows the per-platform strategy parameters that constrain the generation.

TABLE I. PLATFORM CONTENT STRATEGY PARAMETERS

Platform	Max Length	Hashtags	Media Format
LinkedIn	1300	5	Image / carousel
Instagram	600	20	Square / vertical
Twitter	280	3	Image / GIF / video
Facebook	800	3	Image / link preview

The compiled system prompt instructs the model to adhere to the maximum length, include hashtags in the post, and make sure the content is native to the selected platform and not generic. The business modifier adds context like innovation for startups or mission and impact for non-profits. An accompanying user-prompt builder builds the topic, with optional overrides for audience, tone, and call-to-action provided by the caller.

E. Data Curation Pipeline

A curation pipeline is a subgraph of a LangGraph that takes raw material from multiple sources, validates and groups related items, and formats the result for downstream report generation. Twitter data are validated in batch and clustered by content with reflection and re-clustering passes; GitHub trending repositories and Reddit posts are validated via dedicated wrappers while generic URLs are validated individually. AI-newsletter content can be extracted from e-mail or feed sources as well. Verification and grouping are sent parallel via Send primitives. Formatted output is given to the supervisor's report generation stage.

Each verification task is directed into its own parallel branch: general URLs, raw trending repositories, raw Reddit posts and validated tweets are processed in parallel. The content is verified, grouped and, where similar tweet groups are detected, reflected upon and re-grouped to reduce duplication. The final structured output is available to the supervisor for report generation.

1) Verification Nodes

Typed state over focus nodes reduces verification. Tweet verification checks each candidate tweet against eligibility criteria before it is accepted into the pool. Reddit-post verification turns a submission into a canonical string representation, so it can be consistently used by downstream report generation. The link verification process is used to validate external URLs in Tweets or general content to ensure that any links referenced do in fact resolve, and are appropriate, before being passed along. The GitHub validation takes the raw trending-repository payload and validates each repository before adding it to a report. Each node outputs structured, typed results, preserving the guaranties that the graph relies on.

2) Grouping and Reflection

Twitter content is organized by common content to avoid duplicated coverage. A reflection pass is performed to revisit the grouped tweets and find groups that are still too similar. The formatted data is then finalized by marking them for re-grouping. Any similar groups left are merged by the re-grouping node. This reflect-and-merge cycle reduces duplication in the final curated feed, while keeping truly separate items separate.

A similar repurposing subsystem transforms existing content into alternative formats and references a

memory and reflection graph to maintain context across interactions. The deployment of these agents suggests an extensible orchestration model, where new capabilities are added by adding further graph nodes and prompts.

F. Analytics and Database

The analytical data is stored in a SQLite database and the schema defines three tables: `post_analytics`, `platform_metrics` and `generated_content`. The `post_analytics` table records per-post identifiers, platform, content type, content, timestamps, follower, reach, impressions, likes, comments, shares, saves, clicks, an engagement rate and a sentiment label. The `platform_metrics` table contains the total followers and the follower growth per platform and date, and the `generated_content` table contains the created posts and image paths.

If you upload a dataset as CSV through the dashboard it is parsed and normalized to this schema, so you can have whatever column names you want. The ingestion routine supports many aliases for each field (impressions, views, and impression), and calculates the engagement rate as the ratio of total engagement to the effective reach. Sentiment labels are mapped to a canonical positive, negative or neutral value. Rows without an identifier or timestamp are assigned generated values, and a warning list is returned to the caller. To maintain the consistency of the dataset, the rows of the previous table are deleted before ingestion.

An analytics service issues SQL aggregate queries over the ingested rows to compute overview metrics, platform performance, temporal trends, content-type performance, sentiment distribution, top and bottom posts by engagement rate, paginated post listings, and best posting hours. The overview query calculates total reach, impressions, engagement, post count and per-post averages. The engagement rate is calculated as described in (2).

$$\text{Engagement Rate} = \left(\frac{\text{Total Engagement}}{\text{Total Reach}} \right) \times 100$$

G. Insight Synthesis

Insight generation is a structured textual summary of computed analytics, covering overview metrics, platform performance, content-type performance, sentiment distribution, top and lowest performing posts, and best posting times. The summary and an analyst system prompt are fed to the local Qwen model. The prompt asks the model to provide key findings, problems detected and recommendations. It also asks it to ground every statement in the numbers provided and not to make up any statistics. A parser retrieves the three labeled sections from the model output and displays them to front end as structured lists, with a fallback to surface the raw text if the labeled structure does not exist.

H. Social Platform Clients

Automated posting is enabled by a Twitter client and a LinkedIn client. This twitter client supports two types of authentication; user credentials directly or OAuth managed by the Arcade. It offers methods for user authorization, tweet retrieval, tweet search and list, tweet and thread posting, media upload, retweeting, and credentials validation. The read-write client performs media upload, and a text-only mode disables media handling altogether if media credentials are unavailable. The authentication mode is chosen at run-time from configuration flags, either a direct credential-based post, or an interactive OAuth authorization flow.

The LinkedIn client is an example of this pattern, in which it enforces the network's posting restrictions and authentication requirements to publish text and media for professional audiences. The two clients are the main distribution channels for the content that is created, and the curation and analytics capabilities are not tied to a specific platform.

I. Frontend Dashboard

The front end is a single page application built with Vite and React, styled with a professional dark-mode stylesheet. It has three principal views. The dashboard view shows key performance indicators, trend lines for reach and engagement, a bar chart for platform comparison, content type performance, a sentiment pie chart, and tables of top and lowest performing posts, all rendered with the Recharts library. The generation view renders the returned post and optional image, and exposes controls for platform, business type, topic, audience, tone, call-to-action, and image generation. The upload view can consume CSV workbooks and report on the number of rows consumed and columns detected.

J. REST Application Programming Interface

The backend exposes a set of REST endpoints, grouped by concern: The analytics router provides overview metrics, platform performance, temporal trends, content-type performance, sentiment distribution, paginated listings of posts with platform filtering, top and bottom posts, best posting times, and a backwards-compatible dashboard endpoint. The generation router receives a content-generation request with topic, platform, business type and optional audience, tone, call-to-action and image flag, calls the Main Agent graph and returns the generated text with an optional image path. Second endpoint will generate image from prompt, width and height. The dataset router accepts multipart uploads of CSV, XLSX or XLS files, checks the file extension, enforces a 50 megabyte limit, saves the file into an upload directory and reports the number of rows ingested and detected columns and any warning. The insight generation service is called on-demand by the insight router. The health router announces the availability of the backend and its supporting services (Table II).

TABLE II. PRIMARY REST ENDPOINTS

Route	Method	Purpose
/api/analytics/overview	GET	Overview metrics
/api/analytics/platforms	GET	Platform performance
/api/analytics/trends	GET	Reach/engagement trend
/api/analytics/posts	GET	Paginated post listing
/api/analytics/posts/top	GET	Top and bottom posts
/api/generate-content	POST	Generate a post
/api/dataset/upload	POST	Ingest CSV dataset
/api/insights	POST	Generate AI insights
/api/health	GET	Service health status

Validation of generation inputs: missing topic is rejected with 400, invalid business types are normalized to default and graph errors are surfaced as gateway errors. The multipart upload handler limits the extensions that are accepted and enforces the size limit. These validations and structured error objects from the services provide a predictable interface contract to the front end.

IV. RESULTS

The system was put into production and tested end to end. The backend, database and front were built and ran successfully. When the llamafile service was accessible, the analytics service computed the defined metrics from an ingested sample dataset and the generation pipeline produced platform-specific posts from the local Qwen model. In this phase, no accuracy or latency numbers are reported due to the prototype environment and no quantitative performance benchmarks were captured.

A. Content Generation

The generation route, given platform, business type and topic, generated a post structured according to the corresponding strategy and constrained to the maximum length of the platform. For example, switching the selected network from Twitter to LinkedIn changed the tone, hashtag count and structural template that was reflected in the output. This shows the effect of strategy compilation on generation behaviour. The generation request triggers the Main Agent graph which detects the intent, build the strategy, generate the text and when required invoke the local image generator returning the assembled result to the front end.

B. Analytics Computation

Load sample CSV to populate the post_analytics table. The analytics service then calculated total reach, impressions, engagement, engagement rate, per-post averages, follower totals, platform performance, time trends, content type performance, sentiment distribution and best posting times. The dashboard visualised these metrics using the Recharts line, bar and pie charts described above and the paginated post listing returned ranked posts with engagement rates. Figs. 2-5 show representative outputs generated from ingested dataset.

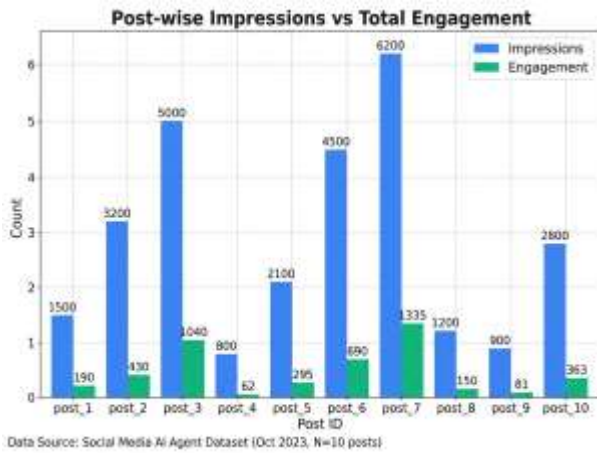


Fig. 2. Daily impressions distribution.

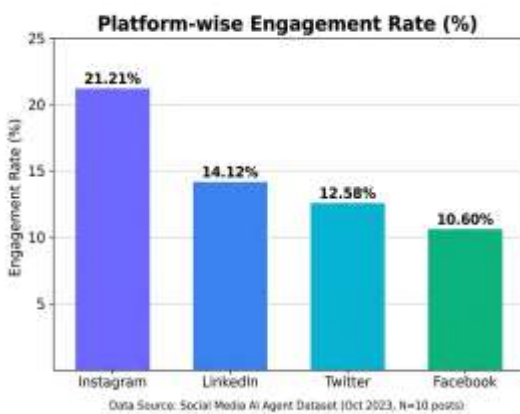


Fig. 3. Platform-wise engagement and reach comparison.

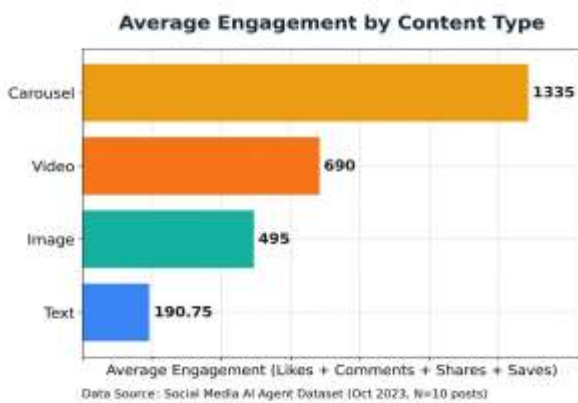


Fig. 4. Average engagement by content type.

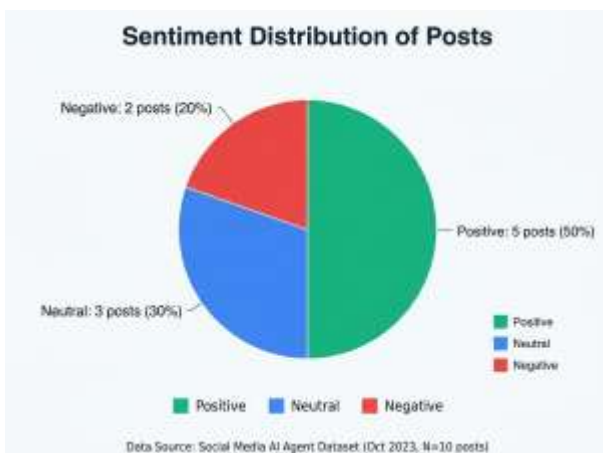


Fig. 5. Sentiment distribution of posts.

C. Insight Generation

The insight service created a textual summary of the calculated analytics and delivered it to the local Qwen model with the help of the analyst prompt. The returned content was parsed into key findings, problems and recommendations, all based on the numbers provided. When the local LLM endpoint was unavailable, the service and dashboard showed a clear, actionable error instead of failing silently.

D. Testing and Validation

The project includes a unit-test suite and a separate integration-test suite with Jest. Unit tests verify link-verification state transitions, posting uploads logic, and scheduling date calculation. The integration tests exercise curation, the twitter client, reddit interactions, supervisor & repurposer graphs, and verification flows. Continuous integration runs project-wide formatting, linting and spelling checks on all pull requests, and has unit tests and daily scheduled integration tests in separate workflows. Table III. Primary automated test categories.

TABLE III. AUTOMATED TEST CATEGORIES

Category	Scope	Execution
Unit tests	Link state, upload, scheduling	On CI
Integration	Curation, clients, graphs	Daily cron
Formatting	Prettier formatting	On CI
Linting	ESLint rules	On CI
Spelling	codespell dictionary	On CI

E. End-to-End Workflow

The whole workflow works as follows. In the generation view a user selects a platform, a business type and a topic and submits the request. The generation pipeline creates a user request, invokes the Main Agent graph, and produces the generated post with the optional generated image. In parallel, a user uploads a CSV analytics dataset through the upload view, the dataset is normalised into SQLite and the dashboard shows the calculated metrics. Click generate insights and the calculated summary is sent to the local model and the returned findings, problems and recommendations are displayed. And last but not least, the platform clients can publish content to Twitter and LinkedIn. Combined, these flows form a closed-loop from data and topic to published, analysed and interpreted social media activity.

V. DISCUSSION

The architecture demonstrates that a locally hosted open-weights model can drive a full content generation and analytics pipeline without any cloud inference dependency. The workflow has been beneficial to represent as an explicit state-machine in LangGraph, since nodes are independently testable, conditional routing cleanly separates generation from analysis, and

subgraphs are reusable in the monolithic Main Agent and the parallel Supervisor. Concurrent synthesis of reports from heterogeneous content sources with fan-out enabled using Send primitives.

A. Design Trade-Offs

No claims can be made about the accuracy of the generation without a dedicated evaluation, as no quantitative benchmarks have been collected in this prototype phase. Deploying a four-billion-parameter model locally avoids per-request cloud costs and privacy issues, but requires hardware and longer inference latency. The Twitter client supports two authentication modes, direct-credential and OAuth-token, but increases the configuration surface area. Similarly, breaking down concerns into graph nodes, services and clients improves maintainability, however at the cost of a larger codebase to navigate.

B. Reliability and Failure Handling

The system degrades gracefully when the local model is unavailable. The inference service returns structured errors, the generation and insight routes translate these into HTTP responses and the dashboard shows an informative message. Dataset ingestion reports column detection and per-row warnings so that partially malformed data can be loaded instead of outright rejected. These design choices lead to a predictable user experience even when the service is only partially failing.

VI. CONCLUSION

In this project we designed and implemented a local-first, agentic social media content generation and analytics system. The platform includes a LangGraph orchestration layer, a locally hosted Qwen large language model, per-platform and per-business content strategies, a supervised multi-source curation graph, SQLite-based analytics computation, an Express REST API and a React dashboard, as well as authenticated posting to Twitter and LinkedIn. The system successfully demonstrates the feasibility of an end-to-end marketing workflow driven by open-weights inference, from ingesting raw datasets to data-grounded, model-generated insights.

Future work includes completing the post-generation graph and its prompt engineering, expanding automated unit coverage to all graph nodes, collecting quantitative generation and insight-quality benchmarks, containerizing the deployment, fixing version-manager inconsistencies in the repo, and hardening the platform clients and authentication flows for production use.

VII. ACKNOWLEDGMENT

The authors express their sincere thanks to Mr. Suresh Kumar C, Asst. Professor for his guidance, encouragement and constant support throughout this work. The authors also thank the institution and the

department for providing necessary facilities and resources to undertake this work.

VIII. REFERENCES

- [1] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention Is All You Need," *Advances in Neural Information Processing Systems*, vol. 30, 2017. doi: 10.48550/arXiv.1706.03762.
- [2] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," in *Proc. NAACL-HLT*, 2019, pp. 4171–4186. doi: 10.18653/v1/N19-1423.
- [3] T. B. Brown et al., "Language Models are Few-Shot Learners," *Advances in Neural Information Processing Systems*, vol. 33, 2020. doi: 10.48550/arXiv.2005.14165.
- [4] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen, "LoRA: Low-Rank Adaptation of Large Language Models," 2021. doi: 10.48550/arXiv.2106.09685.
- [5] T. Dettmers, A. Pagnoni, A. Holtzman, and L. Zettlemoyer, "QLoRA: Efficient Finetuning of Quantized LLMs," 2023. doi: 10.48550/arXiv.2305.14314.
- [6] L. Ouyang et al., "Training Language Models to Follow Instructions with Human Feedback," *Advances in Neural Information Processing Systems*, vol. 35, 2022, pp. 27730–27744. doi: 10.52202/068431-2011.
- [7] Y. Wang, Y. Kordi, S. Mishra, A. Liu, N. A. Smith, D. Khashabi, and H. Hajishirzi, "Self-Instruct: Aligning Language Models with Self-Generated Instructions," in *Proc. 61st Annual Meeting of the Association for Computational Linguistics*, 2023, pp. 13484–13508. doi: 10.18653/v1/2023.acl-long.754.
- [8] R. Rafailov, A. Sharma, E. Mitchell, C. D. Manning, S. Ermon, and C. Finn, "Direct Preference Optimization: Your Language Model is Secretly a Reward Model," *Advances in Neural Information Processing Systems*, vol. 36, 2023. doi: 10.52202/075280-2338.
- [9] A. Yang et al., "Qwen2 Technical Report," 2024. doi: 10.48550/arXiv.2407.10671.
- [10] A. Yang et al., "Qwen2.5 Technical Report," 2024. doi: 10.48550/arXiv.2412.15115.
- [11] H. Touvron et al., "Llama 2: Open Foundation and Fine-Tuned Chat Models," 2023. doi: 10.48550/arXiv.2307.09288.

[12] L. Tunstall et al., “Zephyr: Direct Distillation of LM Alignment,” 2023. doi: 10.48550/arXiv.2310.16944.

[13] B. Batrinca and P. C. Treleaven, “Social Media Analytics: A Survey of Techniques, Tools and Platforms,” *AI & Society*, vol. 30, pp. 89–116, 2015. doi: 10.1007/s00146-014-0549-4.

[14] M. K. Hayat, A. Daud, A. A. Alshdadi, A. Banjar, et al., “Deep Learning for Social Media Analytics: A Survey,” *IEEE Access*, 2019. doi: 10.1109/ACCESS.2019.2905101.